1997-06

Knapsack cuts and explicit-constraint branching for solving integer programs

Appleget, Jeffrey A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

KNAPSACK CUTS AND EXPLICIT-CONSTRAINT BRANCHING FOR SOLVING INTEGER PROGRAMS

by

Jeffrey A. Appleget

June 1997

Thesis Advisor: R. Kevin Wood

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1997	Doctoral Dissertation

4. TITLE AND SUBTITLE
Knapsack Cuts and Explicit-Constraint Branching For Solving Integer Programs

5. FUNDING NUMBERS

6. AUTHOR(S)
Appleget, Jeffrey A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Enhanced solution techniques are developed for solving integer programs (IPs) and mixed-integer programs (MIPs). Previously unsolvable problems can be solved with these new techniques. We develop knapsack cut-finding procedures for minimal cover cuts, and convert existing cut-strengthening theory into practical procedures that lift and tighten violated minimal cover valid inequalities to violated knapsack facets in polynomial time. We find a new class of knapsack cuts called "non-minimal cover cuts" and a method of lifting them called "deficit lifting." Deficit lifting enables all of these cuts to be lifted and tightened to facets as well. Extensions of these techniques enable us to find cuts for elastic knapsack constraints and cuts for non-standard knapsack constraints. We also develop the new technique of "explicit-constraint branching" (ECB). ECB enables the technique of constraint branching to be used on IPs and MIPs that do not have the structure required for known "implicit constraint branching" techniques. When these techniques are applied to 84 randomly generated generalized assignment problems, the combination of knapsack cuts and explicit-constraint branching were able to solve 100% of the problems in under 1000 CPU seconds. Explicit constraint branching alone solved 94%, and knapsack cuts solved 93%. Standard branch and bound alone solved only 38%. The benefits of these techniques are also demonstrated on some real-world generalized assignment and set-partitioning problems.

14. SUBJECT TERMS
Optimization, Integer Programming, Knapsack facets, Constraint branching, Generalized Assignment Problem

15. NUMBER OF PAGES
146

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Approved for public release; distribution is unlimited

KNAPSACK CUTS AND EXPLICIT-CONSTRAINT BRANCHING FOR SOLVING INTEGER PROGRAMS

Jeffrey A. Appleget
Lieutenant Colonel, United States Army
B.S., United States Military Academy, 1979
M.S., Rensselaer Polytechnic Institute, 1989

Submitted in partial fulfillment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH
from the
NAVAL POSTGRADUATE SCHOOL
June, 1997

ABSTRACT

Enhanced solution techniques are developed for solving integer programs (IPs) and mixed-integer programs (MIPs). Previously unsolvable problems can be solved with these new techniques. We develop knapsack cut-finding procedures for minimal cover cuts, and convert existing cut-strengthening theory into practical procedures that lift and tighten violated minimal cover valid inequalities to violated knapsack facets in polynomial time. We find a new class of knapsack cuts called "non-minimal cover cuts" and a method of lifting them called "deficit lifting." Deficit lifting enables all of these cuts to be lifted and tightened to facets as well. Extensions of these techniques enable us to find cuts for elastic knapsack constraints and cuts for non-standard knapsack constraints. We also develop the new technique of "explicit-constraint branching" (ECB). ECB enables the technique of constraint branching to be used on IPs and MIPs that do not have the structure required for known "implicit-constraint branching" techniques. When these techniques are applied to 84 randomly generated generalized assignment problems, the combination of knapsack cuts and explicit-constraint branching were able to solve 100% of the problems in under 1000 CPU seconds. Explicit constraint branching alone solved 94%, and knapsack cuts solved 93%. Standard branch and bound alone solved only 38%. The benefits of these techniques are also demonstrated on some real-world generalized assignment and set-partitioning problems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I want to thank my advisor, Kevin Wood, for taking on this challenge with me. His patience, scholarship and friendship have been instrumental in the accomplishments detailed in this document and my education here at NPS. Every time I talk to Kevin, I realize how much more there is to know in this discipline. Jerry Brown was around when I needed him most, and always had sage advice to offer. His confidence in my abilities was a valuable contribution to my achievements.

Al Washburn chipped in and helped me prepare for my orals, and stepped forward to help proofread this document, despite a heavy teaching load. The breadth of his knowledge is always amazing, and an inspiration to those of us who struggle just to learn one discipline.

Craig Rasmussen is the secret weapon of OR PhD students here at NPS. Craig teaches us how to prove things, and spends countless hours with some of us that need the extra help. His endless patience and willingness to shift his calendar around to accommodate needy (sometimes desperate) students will always be remembered.

Francois Melese was a tremendous help to me as I prepared for my oral exam. His willingness to dive into unfamiliar waters and learn new things is truly inspiring.

There were many others on the faculty here at NPS who contributed to my success here at NPS. While there is not enough space to name them all, Rob Dell and Lyn Whitaker served as my "shadow" advisors. Their advice and encouragement were a far bigger help than they might realize.

The Herculean efforts of Tom Halwachs, master of all things that compute in the OR department, are deeply appreciated. Demonstrating the viability of my research could not have been done without his talents.

Thanks to COL Chris Arney and the Department of Mathematics, US Military Academy, for allowing me to take a semester off and return to Monterey to finish my dissertation. The four additional months I spent here at NPS were supported by a grant from the Faculty Development and Research Fund, Office of the Dean, US Military Academy, and funding from the Department of Mathematical Sciences, US Military Academy at West Point, New

York.

Last, but certainly not least, was the contribution from my family and friends. The faith that my friends and family members had in me went a long way, and their support is truly appreciated. My Mom and Dad instilled in me the work ethic and tenacity that allowed me to persevere, even when things looked grim, and were always there to offer encouragement. My dear wife Donna and wonderful son Wesley did without a husband and father for the better part of four years, and this accomplishment would not have happened without their love and support. You're getting me back now, hopefully not too much the worse for wear. And I count the numerous blessings that I have received from God during this challenge. The Lord answered many prayers over the last four years, restored my spirit and provided illumination during some of the darkest moments.

I. INTRODUCTION

This dissertation develops new solution techniques, and improves existing solution techniques for integer programs (IPs) and mixed-integer programs (MIPs). A linear program (LP) minimizes or maximizes a linear objective function subject to linear constraints. LPs have continuous variables; restricting some of the variables in an LP to be integer results in a MIP; restricting all variables to be integer results in an IP. Mixed-integer and integer programs are used to model many planning problems of the military and industrial world such as production planning, vehicle routing and scheduling, fleet management and weapons system procurement.

In the last few years, commercial solvers have dramatically improved solution times for large LPs, but even modest-sized IPs remain difficult to solve. As an example of comparable difficulty, consider one of our test problems (denoted "DLWRD" in Chapter VI) which is a truck-routing problem in the petroleum industry: This IP has only 89 constraints and 469 variables but cannot be solved in less than 1,000 seconds on a fast IBM workstation using good, off-the-shelf technology; the corresponding LP (formed by allowing the integer variables of the IP to admit continuous solutions) can be solved in a fraction of a second, however. This disparity in solution times indicates that additional research on efficiently solving IPs and MIPs is warranted.

The solution techniques developed in this dissertation are used to solve certain previously unsolved problems. We call the combination of these techniques "composite enumeration." Composite enumeration combines constraint-generation and constraint-branching techniques within a standard (variable-based) branch-and-bound framework. These techniques will be described in greater detail later in this chapter.

Any solution technique for MIPs developed in this dissertation will be valid for (pure) IPs as well. The scope of applicability will be clearly stated for each solution technique described. To avoid repeating "IPs and MIPs," subsequent references to "MIPs" will mean "IPs and MIPs" unless stated otherwise.

A. SOLUTION TECHNIQUES FOR MIPs

1. Introduction

Composite enumeration is the combination of techniques that will be used for solving MIPs. It works within the framework of "branch and bound," one of the two standard solution techniques for MIPs, but it also uses the other standard technique, "cutting planes." Cutting planes are constraints that are generated and then added to the original problem to help solve it. Understanding these two basic techniques is essential to understanding the fundamentals of composite enumeration, so a brief introduction to these techniques is given in this section. Readers familiar with branch and bound and cutting planes may wish to proceed directly to section C where the specific research areas of this dissertation are discussed.

To begin, we consider the following IP (most of the discussion here is valid for MIPs as well as IPs but clarity is enhanced by considering pure IPs only):

$$\begin{array}{rlrcll} z^* & = & \min & \mathbf{cx} & & \\ & & \text{subject to:} & \mathbf{Ax} & = & \mathbf{b} \\ & & & \mathbf{0} \leq \mathbf{x} & \leq & \mathbf{u},\ \mathbf{x} \text{ integer} \end{array} \qquad \text{(I.1)}$$

where $\mathbf{A}$ is an $m \times n$ matrix and the other vectors are commensurately dimensioned. The optimal solution to (I.1) is $\mathbf{x}^*$, and $z^* = \mathbf{cx}^*$. Note that this problem and its LP relaxation can never be unbounded since all variables are bounded above and below. Although we show (I.1) with equality constraints, constraints with senses of $\leq$ and $\geq$ are admitted in IPs as well. If the vector $\mathbf{u}$ is a vector of all 1s, then the IP is a binary IP.

Both standard branch-and-bound and cutting-plane solution procedures begin by solving the "LP relaxation" of the IP. The LP relaxation is simply the IP with the integer restriction on the variables relaxed, which allows the variables to take on appropriately bounded continuous values. (A precise description appears in Chapter II.) The solution to the LP relaxation is used as a starting point for both standard branch-and-bound and cutting-plane solution procedures. If the LP solution is integer, the IP has also been solved. If not, the LP solution still provides useful information. The LP's optimal objective function

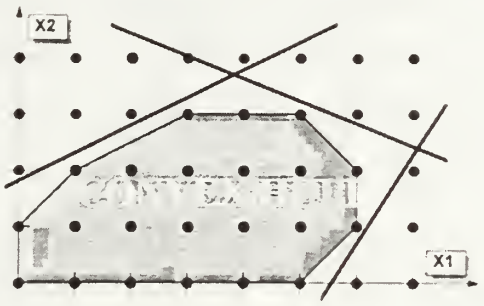

Figure 1. The convex hull of integer solutions.

value is a lower bound on the IP's optimal objective function value, and it is a reasonable assumption (although not always true) that the integer optimal solution will be close to the LP solution.

Before describing branch and bound, cutting planes, and an enhancement to standard branch and bound called "constraint branching," a brief description of the integer polytope is included to introduce terminology and concepts referred to later in this chapter.

2. Description of the Integer Polytope

The feasible region of an IP is a lattice of integer points. The smallest polytope that encompasses all feasible points of the IP is called "the convex hull of integer solutions" or, simply, "the convex hull." The convex hull is a polytope with integer vertices at all corner points. (See Figure 1.) If the constraint set for this polytope were known for an IP, an application of the simplex method to the LP relaxation of the IP would lead to an optimal integer solution (and thus a solution to the IP), because the simplex method proceeds from corner point to corner point, and all corner points of the convex hull are integral. The individual constraints that comprise the minimal constraint set of the convex hull are referred to as "facets." The set of facets for an IP is unique (e.g., Nemhauser and Wolsey, 1988, p. 91).

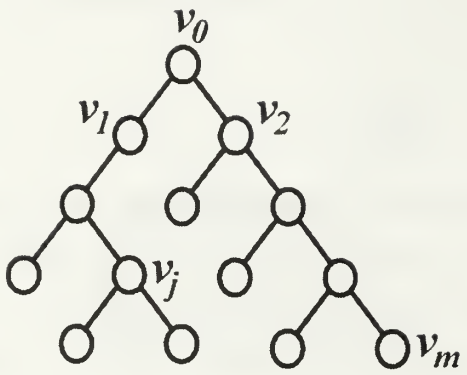

Figure 2. An enumeration tree.

3. Branch and Bound

The following explanation of enumeration trees, branch and bound, and the general branch-and-bound algorithm follows Garfinkel and Nemhauser, (1972, pp. 108-118). The explanation is in the context of (I.1), a minimization problem.

a. Enumeration tree

The branch-and-bound process is best described with an enumeration tree. In an enumeration tree, each node v_j represents a problem to be solved that relates back to the original problem v_0. The feasible region of v_j is the feasible region of v_0 with a restriction added for each branch in the path P_j from v_0 to v_j. Thus, v_j is the same as v_0 with n additional restrictions, assuming P_j has passed through $n-1$ nodes between (and not including) v_0 and v_j. Each node has only one immediate predecessor, but may have many immediate successors. For example, in Figure 2, v_2 has v_0 as its immediate predecessor and has two immediate successors. The following description is of LP-based branch and bound for IPs, but any kind of useful relaxation of an IP may be substituted for the LP relaxation.

Branch and bound begins at v_0 by solving the LP relaxation of v_0, $LP(0)$. If the LP solution is integer, v_0 is solved. If not, branch and bound recursively derives sets of restrictions from LP solutions that define a set of IPs, $v_1, v_2, ..., v_m$ such that the solution to at least one of these IPs solves v_0, if a solution exists. Branch and bound evaluates the

corresponding relaxations $LP(1)$, $LP(2)$, ..., $LP(m)$ to obtain lower bounds used to prove optimality of the solution obtained, or to prove that no solution exists.

The process of "evaluation" at v_j determines if the solution to v_j (if it exists) could solve v_0. The solution to $LP(j)$ determines whether or not any subsequent IPs can be defined that could lead to a better solution than the best integer solution found thus far, called the "incumbent." If no such IPs can be defined, v_j is said to be "fathomed." If these IPs can be defined, "branching" occurs from v_j.

Branching from v_j defines new IPs that are each a different restriction of v_j, at least one of which will solve v_j, if such a solution exists. Of course, an optimal solution to v_j is a feasible solution to v_0. The restrictions that define $v_{j_1}, v_{j_2}, ..., v_{j_k}$, are derived from the solution to $LP(j)$. These nodes are the immediate successors of v_j.

"Node selection" at v_j determines the next node to be evaluated. At v_j, a node must be selected either (a) after the immediate successors of v_j are defined, or (b) when v_j is fathomed. If branching has occurred from v_j, one of v_j's immediate successors is normally selected. If v_j has been fathomed, any other node that has not been fathomed or branched from is selected. Such nodes are said to be "live." Selected nodes must be evaluated, and the process repeats.

v_j is fathomed when it is determined that branching from v_j will not lead to an optimal solution for v_0 or no improvement on the incumbent can be obtained. The three cases for which v_j is fathomed are:

(1) $LP(j)$ is infeasible. If $LP(j)$ is infeasible, then v_j is infeasible.

(2) The solution to $LP(j)$ is integer. If the solution of $LP(j)$ is integer, then any integer solution found by further branching can have no better objective function value than that found at v_j. (The incumbent's objective function value, UB_0, is an upper bound on z^*. Any subsequent integer solution with a lower objective function value than UB_0 becomes the incumbent, and UB_0 is updated.)

(3) The objective function value of $LP(j)$ is greater than or equal to UB_0. If the objective function value of $LP(j)$ is greater than or equal to UB_0, any integer solution found by branching from v_j would have an objective function value greater than (worse than) UB_0.

We examine the branching process next. We consider the problem (I.1), which is rewritten as

$$z^* = \min \ \mathbf{cx} \quad \text{s.t. } \mathbf{x} \in \mathbf{X},$$

where

$$\mathbf{X} = \{\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\}.$$

If we are attempting to find an $\mathbf{x} \in \mathbf{X}$ that minimizes z^*, v_j restricts x to $\mathbf{X}_j \subseteq \mathbf{X}$. The set $\mathbf{X}_j$ is defined as

$$\mathbf{X}_j = \left\{\mathbf{x} | \mathbf{A}^j\mathbf{x} = \mathbf{b}^j, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\right\}$$

where $\mathbf{A}^j\mathbf{x} = \mathbf{b}^j$ is the original constraint set plus, usually, some additional constraints. When branching from a node, the process of "separation" determines which restrictions are applied to $IP(j)$ to form the IPs represented by the immediate successors of v_j. To view how separation might be achieved at v_j, suppose information from the solution to $LP(j)$ suggests that $x_i^* \approx \left\lfloor \frac{u_i}{2} \right\rfloor$. We consider the restrictions $0 \leq x_i \leq \left\lfloor \frac{u_i}{2} \right\rfloor$ or $\left\lfloor \frac{u_i}{2} \right\rfloor + 1 \leq x_i \leq u_i$, and form a set $\mathcal{X}_{j+}$ consisting of two subsets of

$$\begin{aligned} \mathbf{X}_{j_1} &= \left\{\mathbf{x} | \mathbf{A}^j\mathbf{x} = \mathbf{b}^j, 0 \leq \mathbf{x} \leq \mathbf{u}, 0 \leq x_i \leq \left\lfloor \tfrac{u_i}{2} \right\rfloor, \mathbf{x} \text{ integer}\right\} \\ &= \left\{\mathbf{x} | \mathbf{A}^{j_1}\mathbf{x} = \mathbf{b}^{j_1}, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\right\} \end{aligned}$$

and

$$\begin{aligned} \mathbf{X}_{j_2} &= \left\{\mathbf{x} | \mathbf{A}^j\mathbf{x} = \mathbf{b}^j, 0 \leq \mathbf{x} \leq \mathbf{u}, \left\lfloor \tfrac{u_i}{2} \right\rfloor + 1 \leq x_i \leq u_i, \mathbf{x} \text{ integer}\right\} \\ &= \left\{\mathbf{x} | \mathbf{A}^{j_2}\mathbf{x} = \mathbf{b}^{j_2}, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\right\} \end{aligned}$$

so that $\mathbf{X}_{j_1} \cup \mathbf{X}_{j_2} = \mathbf{X}_j$. In this case, as for most IPs, $\mathcal{X}_{j+}$ is a partition of $\mathbf{X}_j$, but a strict partition is not necessary for the branch-and-bound process to converge (e.g., Garfinkel and Nemhauser, 1972, p.113). Although binary partitioning has been described here, partitioning (and any other separation) can allow nodes to have more than two immediate successors.

We have described the foundations of branch and bound and now provide the details of the complete branch-and-bound algorithm. The problem considered is

$$z^* = \min \ \mathbf{cx} \quad \text{s.t. } \mathbf{x} \in \mathbf{X}. \tag{I.2}$$

Let

$$z_j^* = \begin{cases} \mathbf{c}\mathbf{x}_j^* & \text{if } \mathbf{x}_j^* = \mathbf{x} \text{ solves (I.2)} \\ \infty & \text{if } \mathbf{X}_j = \emptyset. \end{cases}$$

At each node v_j, the restricted problem is

$$z_j^* = \min \ \mathbf{c}\mathbf{x} \quad \text{s.t. } \mathbf{x} \in \mathbf{X}_j. \tag{I.3}$$

A lower bound $LB_j \leq z_j^*$ may be calculated by considering some relaxation of (I.3)

$$z_j^o = \min \mathbf{c}\mathbf{x} \quad \text{s.t. } \mathbf{x} \in \mathbf{R}_j \supseteq \mathbf{X}_j \tag{I.4}$$

and letting

$$LB_j = \begin{cases} z_j^o = \mathbf{c}\mathbf{x}_j^o & \text{if } \mathbf{x}_j^o = \mathbf{x} \text{ solves (I.4)} \\ \infty & \text{if } \mathbf{R}_j = \emptyset. \end{cases}$$

An initial upper bound UB_0 can be calculated by finding any $\mathbf{x}' \in \mathbf{X}$, and letting $UB_0 = \mathbf{c}\mathbf{x}'$. Otherwise, UB_0 is initialized to ∞. The feasible regions considered for the LP-based branch-and-bound algorithm are

$$\mathbf{X} = \{\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\},$$

$$\mathbf{X}_j = \left\{\mathbf{x} | \mathbf{A}^j\mathbf{x} = \mathbf{b}^j, 0 \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\right\},$$

and

$$\mathbf{R}_j = \left\{\mathbf{x} | \mathbf{A}^j\mathbf{x} = \mathbf{b}^j, 0 \leq \mathbf{x} \leq \mathbf{u}\right\}.$$

b. A general branch-and-bound algorithm

The following algorithm demonstrates branch and bound for an IP:

Branch-and-Bound Algorithm for IPs

- **Input:** An IP v_0 ($z^* = \min\ \mathbf{cx} \quad \text{s.t. } \mathbf{x} \in \mathbf{X}_0$) where

$$\mathbf{X}_0 = \left\{\mathbf{x} | \mathbf{A}^0\mathbf{x} = \mathbf{b}^0, \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \text{ integer}\right\}.$$

- **Output:** An optimal solution $\mathbf{x}^*$ to v_0, or a message that no solution exists.
- **STEP 1**: (Initialization.) Let $L = \{v_0\}$ be the initial list of live nodes. Let $UB_0 = \infty$, $LB_0 = -\infty$, and $v_j = v_0$, $j = 0$. Go to STEP 2.
- **STEP 2**: (Evaluation.) Solve $LP(j)$. If $\mathbf{R}_j = \emptyset$ (i.e., $LP(j)$ is infeasible), v_j is fathomed (case 1) and $L = L - \{v_j\}$, go to STEP 6. If $LP(j)$ has an optimal solution $\mathbf{x}_j^o$, let $LB_j = z_j^o$ and go to STEP 3.
- **STEP 3**: (Fathoming, case 2.) If $\mathbf{x}_j^o \notin \mathbf{X}_j$ (i.e., $\mathbf{x}_j^o$ is not integer), go to STEP 4. If $\mathbf{x}_j^o \in \mathbf{X}_j$, let $UB_j = z_j^o$, v_j is fathomed and $L = L - \{v_j\}$. If $UB_j < UB_0$, let $UB_0 = UB_j$ and let $\mathbf{x}^I = \mathbf{x}_j^o$, where $\mathbf{x}^I$ is the incumbent. Go to STEP 6.
- **STEP 4**: (Fathoming, case 3.) If $LB_j \geq UB_0$, v_j is fathomed and $L = L - \{v_j\}$. Go to STEP 6. Otherwise, go to STEP 5.
- **STEP 5**: (Branching.) Choose a separation (partition) $\mathcal{X}_{j+}$ that determines the immediate successors of v_j, $\{v_{j_1}, v_{j_2}, ..., v_{j_k}\}$. Update the set of live nodes $L = L - \{v_j\} + \{v_{j_1}, v_{j_2}, ..., v_{j_k}\}$. Go to STEP 6.
- **STEP 6:** (Node selection.) If $L = \emptyset$, go to STEP 7. Otherwise, select a live node and designate it v_j. Go to STEP 2.
- **STEP 7**: (Termination.) If $UB_0 = \infty$, print "The IP has no feasible solution." If $UB_0 < \infty$, the feasible solution that yielded UB_0 is optimal, print the optimal solution $\mathbf{x}^* = \mathbf{x}^I$.

c. An example of branching on variables

The following is an example of branch and bound for an IP where the partitioning is achieved by "branching on variables." This technique uses bounds on variables to partition the feasible region of the IP and its LP relaxation, and is the most common technique used to solve IPs. The solution vector for $LP(j)$ is denoted $\mathbf{x}_j^o = [\widehat{x}_1, \widehat{x}_2, \widehat{x}_3]$.

Consider the following problem:

$$\begin{array}{lllllll} v_0: & z_0^* = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 & & & \\ & & \text{subject to:} & 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & & & & & x_j & \in & \{0,1,2\}, j = 1,2,3. \end{array} \tag{I.5}$$

The LP relaxation of the problem is

$$\begin{array}{lllllll} LP(0): & z_0^o = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 & & & \\ & & \text{subject to:} & 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & & & & & 0 \leq x_j & \leq & 2, j = 1,2,3. \end{array} \tag{I.6}$$

The solution to $LP(0)$ is $\mathrm{x}_0^o = [2,\ 1\frac{1}{2},\ 0]$, $z_0^o = LB_0 = 28$. Because $\widehat{x}_2$ is fractional, and $LB_0 < UB_0$, a partition of $\mathbf{X}_0$ is chosen that eliminates the current fractional value of $\widehat{x}_2$. The IP at each successor node is v_0 restricted by either

$$x_2 \leq 1 \text{ or } x_2 \geq 2.$$

Adding the restriction $x_2 \leq 1$ to v_0 forms v_1, and adding the restriction $x_2 \geq 2$ to v_0 forms v_2. Both nodes are live, and we arbitrarily select v_1 for evaluation. Attempting to solve the LP relaxation of v_1,

$$\begin{array}{lllllll} LP(1): & z_1^o = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 & & & \\ & & \text{subject to:} & 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & & & & x_2 & & \leq & 1 \\ & & & & 0 \leq x_j & & \leq & 2, j = 1,2,3, \end{array} \tag{I.7}$$

we find the feasible region of $LP(1)$ is empty because forcing $x_2 \leq 1$ makes it impossible to satisfy the first constraint. Thus, v_1 is fathomed (case 1). Since v_2 is still live, we select it for evaluation and solve

$$\begin{array}{lllllll} LP(2): & z_2^o = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 & & & \\ & & \text{subject to:} & 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & & & & x_2 & & \geq & 2 \\ & & & & 0 \leq x_j & & \leq & 2, j = 1,2,3. \end{array} \tag{I.8}$$

The solution to $LP(2)$ is $\mathrm{x}_2^o = [1\frac{1}{4},\ 2,\ 0]$, $z_2^o = LB_2 = 30\frac{1}{4}$. Because $\widehat{x}_1$ is fractional, and $LB_2 < UB_0$, a partition of $\mathbf{X}_2$ is chosen to eliminate the current fractional value of $\widehat{x}_1$.

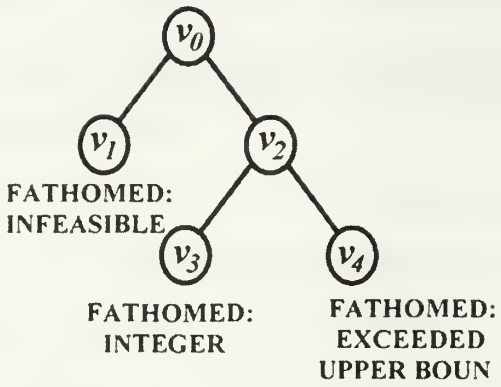

Figure 3. The enumeration tree for the branch-and-bound example.

The IP at each successor node is v_2 restricted by either

$$x_1 \leq 1 \text{ or } x_1 \geq 2.$$

Adding the restriction $x_1 \leq 1$ to v_2 forms v_3, and adding the restriction $x_1 \geq 2$ to v_2 forms v_4. Both nodes are live, and we arbitrarily select v_3 for evaluation. The LP relaxation is

$$\begin{array}{lll} LP(3)\text{: } z_3^o = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 \\ & \text{subject to:} & \begin{array}{rrrcl} 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & x_2 & & \geq & 2 \\ x_1 & & & \leq & 1 \\ & 0 \leq & x_j & \leq & 2, j = 1, 2, 3. \end{array} \end{array} \tag{I.9}$$

The solution to $LP(3)$ is $\mathrm{x}_3^o = [1,\ 2,\ 1]$, $z_3^o = LB_3 = 32$. Because x_3^o is integer, $UB_3 = 32$, and since $UB_0 = \min\{\infty, 32\}$, UB_0 is set to 32 and $\mathrm{x}^I = \mathrm{x}_3^o$, and v_3 is fathomed (case 2). The remaining live node v_4 is selected for evaluation. The LP relaxation is

$$\begin{array}{lll} LP(4)\text{: } z_4^o = & \text{minimize} & 5x_1 + 12x_2 + 3x_3 \\ & \text{subject to:} & \begin{array}{rrrcl} 4x_1 + & 6x_2 + & x_3 & \geq & 17 \\ & x_2 & & \geq & 2 \\ x_1 & & & \geq & 2 \\ & 0 \leq & x_j & \leq & 2, j = 1, 2, 3. \end{array} \end{array} \tag{I.10}$$

The solution to $LP(4)$ is $\mathrm{x}_4^o = [2,\ 2,\ 0]$, $z_4^o = LB_4 = 34$. Because x_4^o is integer, $UB_4 = 34$, and UB_4 is compared with UB_0, which is 32. Since $UB_4 > UB_0$, UB_0 remains the same. Because $LB_4 > UB_0$, v_4 is fathomed (case 3). Since all nodes are now fathomed

(see Figure 3), the integer solution with the lowest objective function value is the optimal solution to v_0. For this problem, the optimal solution was found at v_3 with $z^* = z_3^* = 32$ and $\mathrm{x}^* = \mathrm{x}^I = \mathrm{x}_3^* = [1,\ 2,\ 1]$.

When branch and bound is used on large problems, the usual goal is to obtain an integer solution with an objective function value that is "close enough" to optimality. The use of an "optimality tolerance" precludes branching to find a solution that could only be marginally better than the incumbent, if such a solution exists. The *absolute optimality tolerance* is a value $ABSGAP \geq 0$ specified by the user, and a node is fathomed when it is guaranteed that

$$UB_0 - LB_j < ABSGAP\,.$$

Implementing this amended fathoming rule in the Branch-and-Bound Algorithm for IPs is accomplished by replacing step 4 with:

- **STEP 4′**: (Fathoming, case 3.) Any node j such that $LB_j + ABSGAP \geq UB_0$ is fathomed, go to STEP 6. Otherwise, go to STEP 5.

Solving an IP using this fathoming rule will result in a solution that is guaranteed to be within $ABSGAP$ of the best solution of the problem that could theoretically exist.

The *relative optimality tolerance* is a user-specified percentage $RELGAP \geq 0$; and a node is fathomed when it is guaranteed that

$$(UB_0 - LB_j)/(|LB_j| + \epsilon) < RELGAP\,/100$$

where $\epsilon > 0$. Note that if LB_j and UB_0 have different signs, this procedure will not work. Assuming that LB_j and UB_0 have the same signs, this amended fathoming rule can be accomplished by replacing step 4 with:

- **STEP 4″**: (Fathoming, case 3.) Any node j such that $(UB_0 - LB_j)/(|LB_j| + \epsilon) < RELGAP\,/100$ is fathomed, go to STEP 6. Otherwise, go to STEP 5.

Solving an IP using this fathoming rule will result in a solution that is guaranteed to be

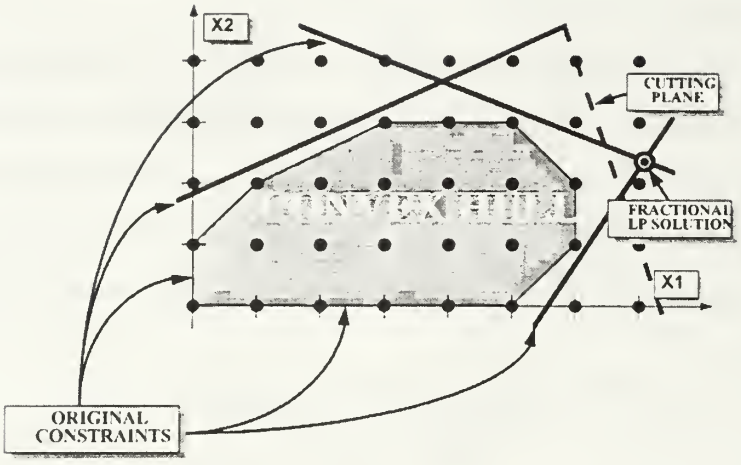

Figure 4. Adding a cutting plane to the constraint set of an IP.

within *RELGAP* percent of the best solution of the problem that could theoretically exist. (A slight error due to ϵ is possible, actually.)

Branch and bound has been the standard way to solve integer programs since the early 1960s, and is credited to Land and Doig (1960). For simplicity, this technique has been demonstrated with an IP, but this process is also applicable to MIPs.

4. Cutting Planes

This discussion focuses on IPs for simplicity, but cutting planes are also used with MIPs.

The cutting-plane technique, more commonly referred to today as "constraint generation," generates additional constraints (cuts) that are appended to the constraint set of the original problem. These constraints must meet two criteria: (a) They must be "valid inequalities" that do not eliminate any of the feasible solutions to the IP when added to the problem, and (b) they must cut away the current fractional (IP-infeasible) solution to the LP relaxation. (See Figure 4.) The goal of a cutting-plane technique is to iteratively cut away fractional solutions to the LP relaxations until the optimal solution to the IP is found

by solving the LP relaxation.

Cutting-plane techniques fall into two general categories, techniques that depend on special structure in the problem ("structure-dependent techniques"), and techniques that can be applied to any IP, independent of the problem structure ("structure-independent techniques").

Examples of structure-dependent techniques include methods for generating valid-inequalities from the set packing polytope (Padberg, 1973), from the knapsack polytope (Padberg 1975), and from the set covering polytope (Cornuéjols and Sassano, 1989). Structure-dependent techniques are relatively easy to implement, but usually cannot guarantee convergence of the cutting-plane technique because they do not derive cuts from the problem's complete constraint set. This dissertation uses cuts derived from the knapsack polytope because knapsack constraints frequently appear in IPs and are fairly easy to find and exploit.

Structure-independent cutting-plane techniques include Gomory cuts (Gomory, 1958), Dantzig cuts (Dantzig, 1959) and the Chvátal-Gomory rounding method (Chvátal,1973). Many of these techniques have finite (convergent) algorithms associated with them. Although initial computational experience with structure-independent techniques was disappointing, more recent research on lift-and-project cuts (Balas, Ceria, and Cornuéjols, 1993) and Chvátal-Gomory cuts (Caprara and Fischetti, 1996) has been more successful. Our initial experience with a structure-independent cutting-plane technique was disappointing, as the computational overhead proved to be significant. Because of this, this dissertation does not consider structure-independent constraint-generation techniques for composite enumeration. This is an area for future research.

On large problems, cutting planes alone have not usually been successful in obtaining integer solutions, but the combination of cutting planes with other techniques continues to produce new solution methodologies for MIPs. Crowder, et al. (1983) combine preprocessing, cutting planes, and a modification of branch and bound to solve large binary IPs. Johnson, et al. (1985) combine preprocessing, coefficient reduction, cutting planes and a modification of branch and bound to solve large-scale binary planning models.

Another combination of techniques called "branch and cut" adds cuts during the

branch-and-bound process. It has been used successfully to solve large MIPs with special structure (Padberg and Rinaldi, 1991, Hoffman and Padberg, 1991, 1993, Balas, Ceria, and Cornuéjols, 1996). The cuts are generated from the solution of an LP at a node within the branch-and-bound enumeration tree. These cuts can be either "globally valid" or "locally valid." Globally valid cuts, such as the polyhedral cuts used by Hoffman and Padberg (1993), are cuts that are valid for any node in the enumeration tree. Locally valid cuts are cuts generated from the restricted LP at a particular node that are only valid for that node and any of its successors. Gomory cuts and knapsack cuts are examples of cuts that are locally valid when generated from the restricted LPs. The knapsack cuts developed in this dissertation could be used in a branch-and-cut framework, but we leave this for future research. We only apply knapsack cuts before the start of branch and bound, where they are globally valid.

Example 1. This example demonstrates a simple cutting-plane technique for MIPs with "knapsack constraints." A pure IP is used for illustration here, but the technique is equally valid for MIPs containing binary knapsack constraints. This technique is formalized in Chapter III, and is the primary cutting-plane technique used in this dissertation. The solution vector for $LP(j)$ is denoted $\mathrm{x}_j^o = [\widehat{x}_1,\ \widehat{x}_2,\ \widehat{x}_3,\ \widehat{x}_4]$, and z_j^o is the objective function value for $LP(j)$.

Consider the problem:

$$\begin{array}{lll} z_0^\star = & \text{maximize} & 10x_1 + 12x_2 + 15x_3 + x_4 \\ & \text{subject to:} & 3x_1 + 4x_2 + 7x_3 + x_4 \leq 12 \\ & & x_j \in \{0,1\}\ j = 1,2,3,4. \end{array} \tag{I.11}$$

This problem is a binary "knapsack problem" (e.g., Garfinkel and Nemhauser, 1972, pp. 13-14). The solution to the LP relaxation of (I.11) is $\widehat{x}_1 = 1$, $\widehat{x}_2 = 1$, $\widehat{x}_3 = \frac{5}{7}$, $\widehat{x}_4 = 0$, $z_0^o = 32\frac{5}{7}$. By considering the knapsack constraint

$$3x_1 + 4x_2 + 7x_3 + x_4 \leq 12, \tag{I.12}$$

and the LP solution of (I.11), it is clear that the three variables that were positive in the solution to the LP relaxation cannot all be set to 1, since this would violate (I.12). An

inequality that expresses this statement is

$$x_1 + x_2 + x_3 \leq 2. \tag{I.13}$$

Because this inequality does not eliminate any valid solutions to (I.11), it is a "valid inequality" for the IP. Because

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 > 2,$$

the valid inequality (I.13) is violated by the fractional solution to the LP relaxation, so (I.13) is also a "cut." By appending (I.13) to (I.11), a new IP is formed:

$$\begin{array}{lllrcl} z_1^* = & \text{maximize} & 10x_1 + 12x_2 + 15x_3 + x_4 & & & \\ & \text{subject to:} & 3x_1 + 4x_2 + 7x_3 + x_4 & \leq & 12 & \\ & & x_1 + x_2 + x_3 & \leq & 2 & \\ & & & x_j & \in & \{0,1\}\ j = 1,2,3,4. \end{array} \tag{I.14}$$

The solution to the first LP relaxation, $\widehat{x}_1 = 1$, $\widehat{x}_2 = 1$, $\widehat{x}_3 = \frac{5}{7}$, $\widehat{x}_4 = 0$, has been cut off by the cutting plane $x_1 + x_2 + x_3 \leq 2$, and is now infeasible to the LP relaxation of (I.14). The solution to the LP relaxation of (I.14) is an integer solution $\widehat{x}_1 = 0$, $\widehat{x}_2 = 1$, $\widehat{x}_3 = 1$, $\widehat{x}_4 = 1$, $z_1^o = z_1^* = 28$, and the problem is solved.

End example 1.

5. Constraint Branching

Since its inception, there has been a quest to improve the process of branch and bound. Branch-and-cut techniques seek to improve the process by using cutting planes. Constraint branching takes a different approach. First, consider variable-based branch and bound for MIPs. It partitions the feasible region of the MIP by branching on single integer variables. For instance, if

$$k < x_j < k + 1$$

for some integer k, the branching choices are

$$x_j \leq k \text{ or } x_j \geq k + 1.$$

This often creates an unbalanced enumeration tree, especially in the case of binary variables.

(We explain the concept of "balance" in the next section.) If we could discover a bounded relationship involving a set of variables, such as

$$k < \sum_j a_j x_j < k + 1,$$

where all a_j as well as k are integer, we could partition the IP's feasible region with respect to the entire expression $\sum_j a_j x_j$. This technique has the potential to create a better balanced enumeration tree. The choice for "constraint branching" might consider the branching choices

$$\sum_j a_j x_j \leq k \text{ or } \sum_j a_j x_j \geq k + 1.$$

Certain applications of this technique have proven to be much more effective than variable-based branch and bound for problems with special structure.

Beale and Tomlin (1973) developed the first application of constraint branching using "special ordered sets." Special ordered sets (SOS) are sets of variables within existing constraints that share a common property. For example, the set of variables in a set-partitioning constraint ($\sum_{j \in J_i} x_j = 1$, x_j binary for all $j \in J_i$) share the Type 1 SOS property that at most one of the variables in the set can be non-zero. Partitioning with SOS is achieved using the solution values from the LP relaxation and the user-implied ordering of the variables. This information is used to define subsets of the variables that determine the restrictions for each new IP formed. The actual restriction is accomplished logically, without adding any new constraints to the MIPs formed at the subsequent nodes. SOS is discussed in more detail in the next section.

The "implicit-constraint-branching" approach of SOS is an efficient technique, since explicitly adding constraints and/or variables to a problem can make the problem more computationally difficult. Although Beale and Tomlin recognized two special ordered sets, most MIP solution software packages today also include a third type of special ordered set (e.g., CPLEX, 1993, pp.71-73). Foster and Ryan (1981) recognize a relationship between sets of variables in overlapping set-partitioning constraints and develop an implicit-constraint-branching technique for problems with this property.

One limitation of implicit-constraint branching is that it can only be used when

special structure exists somewhere in a MIP. Adding additional structure to a MIP allows another type of constraint branching called "explicit-constraint branching" (suggested by Wood, 1994) that has proven useful for certain classes of problems. This technique adds structure, constraints and variables, to facilitate the branch-and-bound process. Our results in Chapter VI show that the increased difficulty resulting from the added structure is often outweighed by a reduction in the size of the enumeration tree. This technique is outlined in section C.3.

a. Implicit-constraint branching

To illustrate why constraint branching might be preferred to variable branching, we discuss an example of implicit-constraint branching with SOS variables.

Assume a MIP, v_0, has q "set-partitioning constraints" of the form

$$\sum_{j \in J_i} x_j = 1,$$

where the x_j are binary variables, J is the set of all variables in the problem, $J_i \subset J$, and $i = 1, ..., q$. We are at v_0 of the branch-and-bound tree, and have solved the LP relaxation of v_0, $LP(0)$. Each set of variables x_j, $j \in J_i$, $i = 1, ..., q$ is an example of an SOS of "Type 1," which is a set of variables of which at most one variable may be non-zero (e.g.,CPLEX, 1993, p.72). We first examine the values of the variables in the solution to $LP(0)$. If all integer variables have integer values, v_0 is solved. Suppose not, and suppose that set-partitioning constraint p has "fractional variables," i.e., variables with fractional LP solution values. The set of variables J_p in the constraint is then partitioned into two disjoint subsets, J_p^k and J_p^{k+1},each containing one or more fractional variables. (The user-specified ordering of the variables also influences the composition of the subsets.) v_0 will be restricted by either

$$\sum_{j \in J_p^{k+1}} x_j = 1 \text{ or } \sum_{j \in J_p^k} x_j = 1.$$

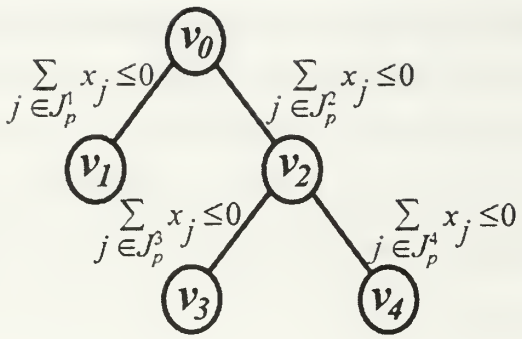

Figure 5. Implicit-constraint branching. Partitioning at v_2 is achieved by forming two disjoint subsets J_p^3 and J_p^4 such that $J_p^1 = J_p^3 \cup J_p^4$.

Those restrictions are equivalent to

$$\sum_{j \in J_p^k} x_j \leq 0 \text{ or } \sum_{j \in J_p^{k+1}} x_j \leq 0. \tag{I.15}$$

Instead of adding the appropriate constraint to v_0, implicit-constraint branching is carried out logically. If from v_0 we select v_1, the first restriction in (I.15) is implicitly added to v_0 by setting $x_j = 0$ for all $j \in J_p^k$. If we select v_2, the second restriction in (I.15) is implicitly added to v_0 by setting $x_j = 0$ for all $j \in J_p^{k+1}$. Because no constraints are actually added to the problem, this is an implicit-constraint-branching technique. (See Figure 5.)

Intuitively, this constraint-branching procedure is preferred to variable-based branch and bound because it eliminates the unbalanced decisions that are made by branching on single binary variables. The variable-based branch-and-bound algorithm must choose a variable to branch on. Setting the variable to 0 is a weak decision, since all the variables in a set-partitioning constraint except one are set to 0 in the final solution anyway. Setting a variable to 1 is a strong decision, because it forces all other variables in the constraint to be set to 0.

The SOS technique makes a choice that moderates the "strength" of the decision made. Branching to a subsequent node restricts all the variables in one subset to 0, and allows any combination of the variables in the other subset to sum to 1. As the branching

progresses, the "strong" decision of setting a particular variable to one is delayed, while unpromising variables are culled out of the unrestricted "sum to 1" subset and placed in the new "set to 0" subset. Once an integer solution is found, other partitions are explored using different subsets, and the process progresses in the same manner. Favorable computational results using SOS are reported by Crowder, et al. (1983) and Hummeltenberg (1984).

We call any enumeration technique for IPs that partitions the feasible region sufficiently to guarantee convergence a "complete" enumeration technique. SOS Type 1 is an example of a complete enumeration technique when all variables fall into at least one set-partitioning constraint. Enumeration techniques that cannot guarantee convergence are called "incomplete." Our basic explicit-constraint-branching technique (outlined in section C.3) is an example of an incomplete enumeration technique: It must revert to variable-based branch and bound on the problem's original variables to guarantee convergence.

b. Generalized branching

"Generalized branching" is a relatively new term. In a generic context, it refers to all branching techniques other than standard variable-based branch and bound. Techniques such as implicit-constraint branching, our explicit-constraint branching, and some branch-and-price techniques (e.g., Sol, 1994) are generalized branching techniques. Jörnsten and Larsson (1988) develop a technique they call "generalized branching" which, as described in Jörnsten and Värbrand (1991), adds branching constraints at nodes of the branch and bound process.

B. ORGANIZATION

This dissertation is organized as follows:

- The remaining sections of this chapter summarize the key results developed in this dissertation. These results include computationally efficient procedures for deriving a violated knapsack facet from an individual knapsack constraint, a new class of knapsack cuts, and a new branching technique called "explicit-constraint branching" that improves branch-and-bound performance.
- Chapter II provides definitions and other preliminary material.

- Chapter III develops two cut-generation techniques for standard knapsack constraints, lifting and tightening techniques that guarantee the creation of stronger cuts from those originally generated, and a new class of knapsack cuts.
- Chapter IV develops a cut-generation technique for elastic knapsack constraints and transformations that allow cut generation for non-standard knapsack constraints.
- Chapter V develops the technique of explicit-constraint branching.
- Chapter VI outlines results of composite enumeration as applied to a series of test problems including modified and standard Generalized Assignment Problems (GAPs).
- Chapter VII summarizes completed research.

C. ENHANCED SOLUTION TECHNIQUES FOR MIPs

1. Introduction

We will show that combining cutting planes derived from individual knapsack constraints with an explicit-constraint branching technique (discussed in section C.3) produces remarkable computational results. While knapsack cuts alone rarely solve previously unsolvable MIPs, our research shows that the combination of knapsack cuts with explicit-constraint branching solves MIPs that cannot be solved in a practical amount of computing time by either of the two techniques applied in isolation. The use of knapsack cuts makes intuitive sense, and they are easy to exploit. The knapsack structure is commonly found in many real-world problems.

2. Cutting Planes from the 0-1 Knapsack Polytope

a. Knapsack facets

A constraint of the form

$$\sum_{j \in J_i} a_j x_j \leq b,$$

where a_j and b are positive integers, x_j are binary variables, and $J_i \subseteq J$ is commonly referred to as a "knapsack constraint." Knapsack constraints are often used to model consumption of limited resources, where b represents the amount of resource available and a_j represents the amount of resource that will be consumed if the variable representing a resource-consuming process x_j is activated by setting $x_j = 1$.

One of the key results of this dissertation is a cutting-plane technique for a knapsack constraint that finds a maximally violated "minimal cover cut," (e.g., Balas and Zemel, 1978) if one exists, and then lifts and tightens the cut to a facet of the knapsack polytope, all in pseudo-polynomial time. We first require a feasible solution to the LP relaxation of the MIP. To be a candidate for a cutting-plane technique, a knapsack constraint must have at least one variable with a fractional LP solution value. For these candidates, we find a maximally violated minimal cover cut, if one exists, in pseudo-polynomial time by using dynamic programming; this requires that the coefficients and the right-hand side of the knapsack constraint be integer. Then, a procedure that combines "lifting" and "tightening" creates a facet from the minimal cover cut in polynomial time. The combination of the two procedures is a technique that finds a maximally violated facet of the knapsack polytope in pseudo-polynomial time. A new cut-finding procedure for a knapsack constraint finds a "non-minimal cover cut" if one exists but no minimal cover cut does. A similar lifting and tightening procedure creates a knapsack facet from this cut as well. Results are extended to elastic knapsack constraints and knapsack constraints with greater-than-or equal-to and equality senses. (Chapter II provides the definitions of "lifting," "tightening," "minimal cover" and "minimal cover cut.")

As with most cutting-plane techniques that rely on limited special structure, these techniques are not convergent. That is, they will not generate cuts that define the convex hull of the MIP in enough detail to solve the MIP. This is true because knapsack facets are derived from individual knapsack constraints, and thus are facets of the knapsack polytope, but are not usually facets of the MIP's polytope. Nonetheless, these knapsack cuts enable other branching techniques to work more quickly and efficiently. Knapsack cuts are derived in Chapter III.

The properties of the knapsack polytope have been explored in the literature

for over 20 years. Our research builds on results found in the seminal papers of Padberg, Balas, and Zemel. Padberg's method for lifting a minimal cover valid inequality to a facet (1975) is a key result that has laid the groundwork for many other knapsack results. From Padberg's result, Balas (1975) develops necessary and sufficient conditions for classes of inequalities to define facets of the knapsack polytope. Balas and Zemel identify the class of all facets associated with minimal covers for a knapsack polytope (1978). Padberg's knapsack result is applied to help solve large-scale binary IPs by Crowder, Johnson, and Padberg (1983). Zemel (1989) develops a method to lift a minimal cover valid inequality to a facet in polynomial time. Research continues today with Boyd's recent papers on Fenchel cutting planes, which are cuts for the knapsack polyhedron (Boyd, 1992, 1994).

b. Computational complexity

Any cutting-plane technique developed for use on practical problems must be computationally tractable. Computational experience has shown knapsack cuts to be useful for reducing variable-based branch and bound enumeration (Crowder et al., 1983), but the additional constraints generated must be more useful than the computational overhead added by the actual process of generating the cuts. The emphasis on computational complexity is demonstrated by the following excerpt from a paper on knapsack cuts:

> ...it is not clear that given a minimal cover from which a violated facet can be generated by lifting and complementing that such a facet can be generated in pseudo-polynomial time (the polynomial time lifting theorem of Zemel guarantees a facet, not a violated facet). (Boyd, 1992)

The lifting and tightening methodology we develop for minimal cover cuts lifts and tightens a minimal cover cut to a violated facet in polynomial time.

c. Cuts for elastic knapsack constraints

An elastic constraint is a constraint from a MIP that has been transformed by the addition of one or more variables that allow penalized violation of the constraint. The penalty is applied by adding each elastic variable to the objective function with some coefficient p that represents the cost per unit of violation. Elastic constraints occur frequently

in LPs and MIPs (e.g., Brown and Graves, 1981). While any constraint can be made elastic, knapsack constraints are prime candidates for "elasticizing." Consider the knapsack constraint

$$\sum_{j \in J_i} a_j x_j \leq b. \tag{I.16}$$

We create an elastic knapsack constraint from (I.16) by adding an elastic variable $z \geq 0$:

$$\sum_{j \in J_i} a_j x_j - z \leq b.$$

The variable z represents the additional units of resource that can be used above and beyond the basic limit of b if an appropriate (linear) penalty is paid. The variable z may or may not be explicitly bounded.

Our research extends the basic knapsack cut results to derive cuts from elastic knapsack constraints. This new result (Chapter III) allows the use of knapsack cutting planes on the Generalized Generalized Assignment Problem (GGAP), an extension of the GAP that uses elastic knapsack constraints. The GGAP is described in Chapter II.

3. Explicit-Constraint Branching

"Explicit-constraint branching" (ECB) is a new technique that allows the benefits of constraint branching for problems lacking the special structure required of known implicit-constraint-branching techniques. This new technique is remarkable because the combination of variable-based branch and bound and ECB solves MIPs that variable-based branch and bound alone cannot solve. When knapsack cuts are added to this combination, the performance is often better yet.

The constraints and general integer variables that ECB adds to a MIP facilitate variable-based branch and bound. This may sound counterintuitive, but a simple example illustrates why this technique has merit.

Example 2. Consider the binary IP (the problem is trivial, but serves to highlight the benefits of ECB on even small problems)

$$\begin{array}{lll} \text{maximize} & \sum_{j \in J} x_j & \\ \text{subject to:} & \sum_{j \in J} 2x_j & \leq \; 2\left\lfloor \frac{|J|}{2} \right\rfloor + 1 \\ & x_j & \in \; \{0,1\} \; \forall j \in J. \end{array}$$

The LP relaxation has an extreme point solution with $\left\lfloor \frac{|J|}{2} \right\rfloor$ of the variables equal to 1, variable equal to .5, and the other $|J| - \left\lfloor \frac{|J|}{2} \right\rfloor - 1$ variables equal to 0. Variable-based branch and bound forms a partition based on the one fractional variable, which we designate as x_f, and derives the restrictions

$$x_f \leq 0 \text{ or } x_f \geq 1.$$

Each subsequent LP relaxation solved at a node of the enumeration tree has one of the remaining unfixed variables fractional, until the last variable is fixed by branch and bound, and an integer solution is finally obtained. Although this integer solution is an optimal solution, the bounding information is not sufficient to fathom other live nodes, so another live node is selected, and after a similar lengthy path around the enumeration tree fixing fractional variables at each node, another (alternate) optimal solution is found. This exhaustive process continues until all $\binom{|J|}{\lfloor \frac{|J|}{2} \rfloor}$ alternate optimal solutions are enumerated.

We employ constraint branching on this problem to reduce the number of nodes branch and bound must enumerate. By adding an additional variable y, called a "branching variable," and a constraint (redundant to the IP) of the form

$$\sum_{j \in J} x_j - y = 0,$$

we form the modified problem

$$\begin{array}{lllll} \text{maximize} & \sum_{j \in J} x_j & & & \\ \text{subject to:} & \sum_{j \in J} 2x_j & & \leq & 2\left\lfloor \frac{|J|}{2} \right\rfloor + 1 \\ & \sum_{j \in J} x_j & -y & = & 0 \\ & x_j & & \in & \{0,1\} \; \forall \, j \in J \\ & & y & \in & \{0,1,2,...,|J|\}. \end{array}$$

The LP relaxation solves exactly as before for the x_j, and the branching variable y is equal to

$\left\lfloor \frac{|J|}{2} \right\rfloor + .5$. The variable-based branch and bound partition based on y derives the restrictions

$$y \leq \left\lfloor \frac{|J|}{2} \right\rfloor \text{ or } y \geq \left\lfloor \frac{|J|}{2} \right\rfloor + 1.$$

The second restriction is infeasible, and the first yields an LP relaxation with optimal extreme points. Variable-based branch and bound, using the simplex algorithm to solve the LP relaxations, requires only three nodes to solve this problem.

End Example 2.

In general, the application of explicit-constraint branching adds a constraint

$$\sum_{j \in J'} x_j - y = 0$$

where $J' \subseteq J$, and J is the set of indices of all integer variables of the problem. ECB constraints can be added for any set of integer variables of a MIP, but the sets of variables used to form ECB constraints must be intelligently chosen if computational improvements are to be realized. ECB constraints are discussed in Chapter V.

The next chapter defines the key terms and introduces the notation that will be used throughout the rest of the dissertation.

II. PRELIMINARIES

This chapter defines key terms used in subsequent chapters. This chapter also gives formulations for the IPs that are used for computational testing of the solution techniques developed in the dissertation.

A. DEFINITIONS

1. General

The notation in this section follows Balas and Zemel (1978). Most of the definitions are standard, but a good reference for them is Nemhauser and Wolsey (1988).

- The *knapsack problem* is a binary integer program that maximizes the sum of the utility of the items that can be carried in a knapsack with limited carrying capacity. For the purpose of illustration, it is assumed that there is a single constraint on the total weight that the knapsack can carry. The problem can be formulated as:

  Indices:

  $j \in N$ items that can be placed in the in the knapsack.

  Given data:

  c_j utility of item j.
  a_j weight of item j.
  b total weight the knapsack can carry.

  Decision variables:

  x_j 1 if item j is placed in the knapsack; 0 otherwise.

  Formulation:

$$\begin{array}{lrcl} \text{maximize} & \sum\limits_{j \in N} c_j x_j & & \\ \text{subject to:} & \sum\limits_{j \in N} a_j x_j & \leq & b \\ & x_j & \in & \{0,1\} \ \forall \ j \in N. \end{array} \tag{II.1}$$

NOTE: The version of the knapsack problem where the x_j are general integer variables is not considered in this dissertation.

- The *LP relaxation* of an integer program $IP(0)$ is an LP, $LP(0)$, which is identical to $IP(0)$ except that integer variables are replaced with appropriately bounded

continuous variables. For example, the LP relaxation of the knapsack problem (II.1) is

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in N} c_j x_j \\ \text{subject to:} \quad & \sum_{j \in N} a_j x_j \leq b \\ & 0 \leq x_j \leq 1 \; \forall \, j \in N. \end{aligned}$$

- The *knapsack inequality* is

$$\sum_{j \in N} a_j x_j \leq b, \tag{II.2}$$

where a_j and b are positive integers, x_j are binary variables, and $N = \{1, \ldots, n\}$.

- The *integer polytope* P associated with a general integer program is the convex hull of points that satisfy

$$P = conv\left\{\mathbf{x} \in Z^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{D}\mathbf{x} = \mathbf{d}\right\}, \tag{II.3}$$

where $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ represents all inequalities, and $\mathbf{D}\mathbf{x} = \mathbf{d}$ represents all equalities in the constraint set of the general IP (see Figure 1).

- The points $\mathbf{x}^1, \ldots, \mathbf{x}^k \in R^n$ are *linearly independent* if the unique solution of $\sum_{i=1}^{k} \lambda_i \mathbf{x}^i = 0$ is $\lambda_i = 0$ for $i = 1, \ldots, k$.

- The points $\mathbf{x}^1, \ldots, \mathbf{x}^k \in R^n$ are *affinely independent* if the unique solution of $\sum_{i=1}^{k} \lambda_i \mathbf{x}^i = 0$, $\sum_{i=1}^{k} \lambda_i = 0$ is $\lambda_i = 0$ for $i = 1, \ldots, k$.

- The following statements are equivalent:
  a. $\mathbf{x}^1, \ldots, \mathbf{x}^k \in R^n$ are affinely independent.
  b. $\mathbf{x}^2 - \mathbf{x}^1, \mathbf{x}^3 - \mathbf{x}^1, \ldots, \mathbf{x}^k - \mathbf{x}^1$ are linearly independent.

- The *dimension* of a polytope P is k if the maximum number of affinely independent points in P is $k + 1$.

- The *knapsack polytope* P associated with (II.2) is the convex hull of 0-1 points that satisfy (II.2):

$$P = conv\left\{\mathbf{x} \in \{0,1\}^n \mid \sum_{j \in N} a_j x_j \leq b\right\}. \tag{II.4}$$

(This is just a special case of (II.3) where the variables are binary and the constraint set consists of one knapsack constraint.)

Assumption 1: It is assumed that the dimension for the knapsack polytope P is

$d = n$, which is true if and only if $a_j \leq b, \forall\, j \in N$ (Balas and Zemel, 1978).

- A *facet* of the integer polytope P (II.4) is an inequality of the form

$$\sum_{j \in N} \omega_j x_j \leq \omega_0 \tag{II.5}$$

that is satisfied by every $\mathrm{x} \in P$, and satisfied at equality by exactly d affinely independent points $\mathrm{x} \in P$, where d is the dimension of P.

- A *valid inequality* for an IP is an inequality of the form

$$\sum_{j \in S \subseteq N} \omega_j x_j \leq \omega_0 \tag{II.6}$$

which, when added to the constraint set of the IP, does not exclude any feasible solution to the IP.

- A *cut* for an IP is a valid inequality that, when added to the IP, eliminates some fractional solution $\widehat{\mathrm{x}}$ to the LP relaxation of the IP. The valid inequality (II.6) is also a cut if

$$\sum_{j \in S} \omega_j \widehat{x}_j > \omega_0.$$

(Cuts are also referred to as "violated valid inequalities," where the violation is $\sum_{j \in S} \omega_j \widehat{x}_j - \omega_0$.)

2. Lifting and Tightening Valid Inequalities

The two processes of lifting and tightening a valid inequality are used throughout the literature on constraint generation, but their definitions often differ between papers. For the purposes of this dissertation, we will make a clear differentiation between the two terms.

- *Lifting* is the process of extending a valid inequality for an IP to include as many additional variables as possible without eliminating any feasible integer solutions. Every variable that is added to (lifted into) the valid inequality may enable the extended valid inequality to eliminate more of the feasible region associated with the LP relaxation of the IP. For example, suppose that by some means an initial valid inequality

$$x_1 + x_2 + x_3 \leq 2 \tag{II.7}$$

is obtained for the following knapsack problem:

$$\begin{array}{lrcl} \text{maximize } 10x_1 + 12x_2 + 15x_3 + 9x_4 & & & \\ \text{subject to: } 3x_1 + 4x_2 + 7x_3 + 10x_4 & \leq & 12 & \\ x_j & \in & \{0,1\}\,,\ j = 1,2,3,4. & \end{array}$$

The valid inequality (II.7) indicates that at most two of the variables x_j, $j = 1, 2, 3$, may be set to 1 without violating the knapsack constraint. One lifting process (called "simple lifting" in this dissertation) extends the valid inequality to include all variables that have constraint coefficients greater than or equal to the largest constraint coefficient of the variables in the valid inequality. Since the coefficient on x_4 is greater than the maximum of the coefficients on x_j, $j = 1, 2, 3$, the valid inequality (II.7) may be lifted (extended) by including the variable x_4 :

$$x_1 + x_2 + x_3 + x_4 \leq 2.$$

- The process of *tightening* a less-than-or-equal-to valid inequality increases the coefficients of the variables from their current value to a higher integer value if it can be done without eliminating any feasible integer solutions. A continuation of the lifting example demonstrates this, using the valid inequality

$$x_1 + x_2 + x_3 + x_4 \leq 2.$$

If x_4 is set to 1, no other variable in the valid inequality can be set to 1 without violating the knapsack constraint. Thus, x_4 can be assigned a coefficient of 2 without excluding any valid solutions, and the tightened valid inequality is

$$x_1 + x_2 + x_3 + 2x_4 \leq 2.$$

3. Covers and Cover Cuts

All definitions in this section refer to the knapsack inequality (II.2) and the associated knapsack polytope P (II.4).

- A *cover* is a set $S \subseteq N$ such that

$$\sum_{j \in S} a_j > b.$$

- A *minimal cover* S is a cover such that

$$\sum_{j \in S - \{i\}} a_j \leq b, \ \forall \, i \in S.$$

- A *minimal cover valid inequality* is a valid inequality that corresponds to the minimal cover S :

$$\sum_{j \in S} x_j \leq |S| - 1. \tag{II.8}$$

- The *extension of* S *(to* N*)* is defined as $E(S) = S \cup \widetilde{S}$, where

$$\widetilde{S} = \{ i \in N - S \mid a_i \geq a_j \ \forall \ j \ \in S \} .$$

- An *extended minimal cover valid inequality* is a valid inequality that corresponds to extension of S :

$$\sum_{j \in E(S)} x_j \leq |S| - 1.$$

- A *minimal cover cut* is a minimal cover valid inequality for which

$$\sum_{j \in S} \widehat{x}_j > |S| - 1 \tag{II.9}$$

where $\widehat{x}_j$ is the jth component of $\widehat{\mathrm{x}}$, the solution vector of the LP relaxation of the MIP.

- A *lifted minimal cover cut* is a minimal cover valid inequality for the minimal cover S extended from (II.9) for which

$$\sum_{j \in E(S)} \widehat{x}_j > |S| - 1.$$

- A *strong cover* S' is a minimal cover for which either

$$E(S') = N$$

or

$$\sum_{j \in S' - \{j_{\max}\}} a_j + a_{i_{\max}} \leq b$$

where

$$j_{\max} = \underset{j \in S'}{\operatorname{argmax}} \, a_j$$

and

$$a_{i_{\max}} = \max_{i \in N - E(S')} a_i.$$

- A *tight strong cover* is a strong cover S' such that

$$\sum_{j \in S'} a_j - b = 1.$$

- A *strong cover valid inequality* is a minimal cover valid inequality (II.8) based on the strong cover S':

$$\sum_{j \in S'} x_j \leq |S'| - 1.$$

- A *strong cover cut* is a lifted strong cover valid inequality:

$$\sum_{j \in E(S')} x_j \leq |S'| - 1.$$

  Note that the strong cover cut must be a lifted strong cover valid inequality to ensure the valid inequality is violated. (This is explained in more detail in Chapter III.)

B. TEST PROBLEMS

This section gives the formulations for the test problems that will be used to evaluate the effectiveness of the computational techniques developed in this dissertation.

1. The Generalized Assignment Problem

One type of IP used for testing is the Generalized Assignment Problem (GAP). This problem arises in a number of contexts (e.g., Amini and Racer 1994, Ross and Soland 1975) but is described here as a minimum cost assignment of orders to trucks. Each order must be delivered, and the number of orders any truck can deliver is constrained by the amount of time the truck has available to make deliveries. It is assumed that deliveries are made from a single depot and that each order o requires one out-and-back trip of known duration. Any order may remain undelivered although a large penalty cost will be incurred if this happens. For notational convenience, each non-delivery is modeled as a delivery by a high-cost phantom truck t' that could deliver all orders by itself. Actual data from the petroleum industry and a set of randomly generated GAPs will be used for tests. All data is integer

except for costs and penalties.

Indices:

$o \in O$ is the set of orders to be delivered,
$t \in T$ is the set of trucks that can make the deliveries,
O_t is the set of orders that truck t is capable of delivering,
T_o is the set of trucks with which order o can be delivered.

Given data:

c_{ot} is the cost of delivering order o with truck t, and
h_{ot} are the hours (in tenths) required by truck t to deliver order o,
H_t are the hours (in tenths) available on truck t, $H_{t'} \equiv \infty$.

Decision variables:

x_{ot} is 1 if order o is delivered by truck t, and is 0 otherwise.

Formulation:

$$\begin{aligned}
\text{minimize} \quad & \sum_{o \in O} \sum_{t \in T} c_{ot} x_{ot} & & & & & \\
\text{subject to:} \quad & \sum_{t \in T_o} x_{ot} & = & 1 & \forall\, o \in O & & \text{(ORDERS)} \\
& \sum_{o \in O_t} h_{ot} x_{ot} & \leq & H_t & \forall\, t \in T & & \text{(TRUCK-HOURS)} \\
& x_{ot} & \in & \{0,1\} & \forall\, o \in O,\ t \in T_o. & &
\end{aligned}$$

2. The Generalized Generalized Assignment Problem

This dissertation also considers the generalization of the GAP (GGAP) with elastic truck hour constraints. These constraints allow penalized overtime on each truck.

Indices: Same as GAP.

Given data: Same as GAP with these additions:

H_t is the maximum number of regular time hours (in tenths) that truck t may operate,
H_t^{δ} is the maximum number of overtime hours (in tenths) that truck t may operate,

Decision variables: Same as GAP with these additions:

y_t^+ overtime hours (in tenths) for truck t

Formulation:

$$
\begin{array}{lrcll}
\text{minimize} & \sum_{o \in O} \sum_{t \in T} c_{ot} x_{ot} + \sum_{t \in T} p_t^+ z_t^+ & & & \\
\text{subject to:} & \sum_{t \in T_o} x_{ot} & = & 1 & \forall\, o \in O \\
& \sum_{o \in O_t} h_{ot} x_{ot} - z_t^+ & \leq & H_t & \forall\, t \in T \\
& x_{ot} & \in & \{0,1\} & \forall\, o \in O, \\
& z_t^+ & \in & \left\{0,1,...,H_t^{\delta}\right\} & \forall\, t \in T
\end{array}
$$

Note that two-sided elastic constraints can also be used to penalize under-utilization of trucks.

III. KNAPSACK CUTS

We use knapsack cutting planes in composite enumeration because knapsack constraints occur often in IPs, they are relatively easy to work with, and their valid inequalities and cuts have been extensively studied in the literature. Despite extensive study, effective and efficient procedures to find and apply knapsack cuts have been lacking. In this chapter, we develop new techniques for finding and lifting cuts from individual knapsack constraints. We describe an algorithm that finds a maximally violated minimal cover cut for a knapsack constraint if such a cut exists, an algorithm that lifts a minimal cover cut to a lifted minimal cover cut and we develop a new lifting procedure called "interior lifting" that lifts a lifted minimal cover cut to a strong cover cut (a lifted strong cover valid inequality). We adapt polynomial-time facet-finding algorithms to tighten strong cover cuts to violated facets, identifying conditions and creating procedures that streamline the facet-finding process.

We identify a new type of knapsack cut, the "non-minimal cover cut," a cut that cannot be found by solving the traditional minimal cover separation problem (e.g., Crowder, et al. 1983). We develop non-minimal cover cut-finding procedures and a new lifting procedure called "deficit lifting." Deficit lifting creates a violated extended minimal cover valid inequality from a non-minimal cover cut, if necessary. All non-minimal cover cuts can be lifted and tightened to facets in polynomial time. The complete procedure of cut-finding, lifting and tightening for either a minimal or non-minimal cover cut produces a violated facet in pseudo-polynomial time.

A. FINDING A MINIMAL COVER CUT

All cutting planes for MIPs must meet two criteria: (a) They must be valid inequalities that do not eliminate any of the feasible solutions to the MIP when added to the problem, and (b) they must cut away $\hat{x}$, the current fractional (MIP-infeasible) solution to the LP relaxation. We focus on the minimal cover valid inequality (MCVI) because it has been the foundation of many of the important results concerning the knapsack polytope. In particular,

an MCVI can always be lifted and tightened to a facet of its associated knapsack polytope. Because we seek an MCVI that is also a cut, we want a violated MCVI which we call a "minimal cover cut" (MCC). Two conditions must be satisfied before attempting to find an MCC. First, the solution to the LP relaxation $\hat{x}$ must be fractional. Second, the knapsack constraint that we will generate cuts from must have at least one $\hat{x}_j$ fractional. If those two conditions are met, we attempt to find an MCC for the candidate constraint. Because we want any facet generated through our lifting and tightening algorithms to have the largest violation possible, we begin the facet-generation process by finding an MCC that is maximally violated, and if found, call it a "maximally violated minimal cover cut" (XVMCC). We describe the XVMCC-finding problem next, and outline an algorithm for its solution.

1. Maximally Violated Minimal Cover Cut-Finding Problem

We solve the XVMCC-finding problem with dynamic programming, which requires that the knapsack constraint have integral data. Before formulating the cut-finding problem, we review the pertinent definitions.

Recall that the knapsack constraint is defined as

$$\sum_{j \in N} a_j x_j \leq b, \tag{III.1}$$

where a_j and b are positive integers, $a_j \leq b \; \forall \; j \in N$, x_j are binary variables, and $N = \{1, ..., n\}$. A cover for (III.1) is a set $S \subseteq N$ such that

$$\sum_{j \in S} a_j > b, \tag{III.2}$$

and a minimal cover S is a cover such that

$$\sum_{j \in S - \{i\}} a_j \leq b, \; \forall \, i \in S. \tag{III.3}$$

The MCVI is written as

$$\sum_{j \in S} x_j \leq |S| - 1.$$

2. Cut-Finding Problem Formulation

In order to find an XVMCC, we must identify a knapsack constraint of the MIP that has at least one variable with a fractional LP solution value $\widehat{x}_j$. Once a suitable constraint is found, we solve the following cut-finding problem:

Indices:

j	$\in N = \{1, 2, 3, ...n\}$ variable index for the knapsack constraint
N^+	$= \{\, j \in N \mid \hat{x}_j > 0\}$ where $\hat{\mathrm{x}}$ is the solution of the LP relaxation of the MIP

Given data:

$\hat{x}_j$	value for x_j in a solution of the LP relaxation of the MIP
a_j	knapsack constraint coefficient of x_j
b	right-hand side of the knapsack constraint

Decision variables:

h_j	1, if x_j is placed in the minimal cover inequality; 0 otherwise
r	a general integer variable that becomes the right-hand side of the XVMCC

Formulation:

$$\text{maximize} \sum_{j \in N^+} \hat{x}_j h_j - r \tag{III.4a}$$

$$\text{subject to:} \sum_{j \in N^+} a_j h_j \geq b + 1 \tag{III.4b}$$

$$\sum_{j \in N^+} a_j h_j \leq b + a_{\min} \tag{III.4c}$$

$$\sum_{j \in N^+} h_j - r = 1 \tag{III.4d}$$

$$h_j \in \{0, 1\} \ \forall\, j \in N^+$$

$$r \in \left\{0, 1, 2, ..., \left|N^+\right| - 1\right\},$$

where $a_{\min} = \min_{j \in N^+} a_j$.

The following list explains the relationship of the constraints to the minimal cover cut that we seek.

(1) Constraint (III.4b) ensures that the right-hand side of the constraint is covered

as required by (III.2):

$$\sum_{j \in N^+} a_j h_j \geq b + 1 \Longleftrightarrow \sum_{j \in N^+} a_j h_j > b.$$

(2) Constraint (III.4c) ensures that the cover is minimal as required by (III.3). If the right-hand side is no longer covered when the element with the minimum coefficient in the cover is removed, it will not be covered if any other $a_j \geq a_{\min}$, $j \in N^+$ is removed:

$$\sum_{j \in N^+} a_j h_j - a_{\min} \leq b \Longleftrightarrow \sum_{j \in N^+} a_j h_j \leq b + a_{\min}.$$

where $a_{\min} = \min_{j \in N^+} a_j$. To ensure that all minimal cover cuts are found, we actually solve the cut-finding problem $|N^+|$ times with each $a_j = a_{\min}$, $j \in N^+$ and requiring $h_j \equiv 0$ for every $a_j < a_{\min}$. (Actually, we need to solve the cut-finding problem only $|N^+| - p$ times, where $|N^+| - p$ is the number of distinct a_j, $j \in N^+$.)

(3) Constraint (III.4d) sets the right-hand side r of the prospective cut:

$$r = \sum_{j \in N^+} h_j - 1 = |S| - 1.$$

(4) A violated MCVI has been found if

$$\sum_{j \in N^+} \hat{x}_j h_j^* - r^* > 0,$$

where $\mathbf{h}^*$ and r^* solve the cut-finding problem.

3. Dynamic-Programming Reformulation

We reformulate the cut-finding problem in order to solve it with dynamic programming. We can combine (III.4a) and (III.4d) by solving (III.4d) for r, and substituting for r in (III.4a). Also, (III.4b) and (III.4c) can be combined into a two-sided constraint. Thus, the actual cut-finding problem we solve is:

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in N^+} (\hat{x}_j - 1) h_j \\ \text{subject to:} \quad b+1 \leq & \sum_{j \in N^+} a_j h_j \leq b + a_{\min} \\ & h_j \in \{0,1\} \ \forall \, j \in N^+, \end{aligned} \tag{III.5}$$

where the problem is solved $|N^+|$ times with each $a_j = a_{\min}$, $j \in N^+$ and requiring $h_j \equiv 0$ for every $a_j < a_{\min}$. A cut is found if the objective function value is greater than -1. Note that this formulation is similar to the constraint identification problem of Crowder et al. (1983) except that our requirement for integral data allows us some simplifications.

The reformulation (III.5) can be solved via this standard dynamic programming recursion:

DP Recursion 1

Initial conditions:
$d_0(0) = 0$
$d_0(a') = -\infty$ for $a' = 1, ..., b + a_{\min}$
$d_k(a') = -\infty$ for all k, $a' < 0$

Recursion:
$d_k(a') = \max \{d_{k-1}(a'), d_{k-1}(a' - a_k) + (\widehat{x}_k - 1)\}$

for $k = 1, ..., |N^+|$; $a' = 0, ..., b + a_{\min}$.

The solution to the problem is

$$v^* = \max_{b+1 \leq a' \leq b+a_i} d_{|N^+|}(a').$$

If $v^* > -1$, then an XVMCC has been found, and the corresponding minimal cover variables and the value of r^* can be recovered through auxiliary data structures within the dynamic programming algorithm.

B. LIFTING A MINIMAL COVER CUT TO A STRONG COVER CUT

In order to obtain a violated facet of the knapsack polytope, we begin with an XVMCC, and apply two lifting processes and one of two tightening processes. This section describes the two lifting processes which create a strong cover cut from a minimal cover cut.

1. Simple Lifting

Simple lifting adds variables to the original minimal cover cut, possibly enabling more IP-infeasible fractional solutions to be eliminated from subsequent solutions of the LP relaxation of the MIP. The resulting strengthened minimal cover cut is called a "lifted minimal cover cut" (LMCC). This lifting process can be performed on any MCVI, and is accomplished by forming the extension of the minimal cover S using the following procedure. The justification follows the procedure statement. (This procedure is very simple, but is stated formally for later reference.)

PROCEDURE 1 (Simple lifting algorithm): This algorithm takes the coefficients of the knapsack constraint with index set N, and the minimal cover S and returns the index set $E(S)$, the extension of the minimal cover.

Input: The index set of the cover S, and the vector of constraint coefficients a_j, $j \in N$.

Output: $E(S)$, the extension of the minimal cover S.

Begin

$a_{j_{\max}} = \max_{j \in S} a_j$

$\widetilde{S} = \{i \in N - S | a_i \geq a_{j_{\max}}\}$

$E(S) = S \cup \widetilde{S}$

Return $E(S)$

End

Given $E(S)$ from Procedure 1, the LMCC is

$$\sum_{j \in E(S)} x_j \leq |S| - 1. \tag{III.6}$$

Note that because $S \subseteq E(S)$, and the right-hand side of the original MCC is the right-hand side of (III.6),

$$\sum_{j \in E(S)} \widehat{x}_j > |S| - 1,$$

and (III.6) is a cut.

Justification for Procedure 1:

For the MCC's minimal cover S

$$\sum_{j \in S} a_j > b,$$

$$\sum_{j \in S - \{j_k\}} a_j \leq b, \ \forall \ j_k \in S,$$

and the MCC is

$$\sum_{j \in S} x_j \leq |S| - 1.$$

If we add any x_i, $i \in N - S$ with $a_i \geq \max_{j \in S} a_j$, there still can be no more than $|S| - 1$ variables x_j that can be set to 1, and thus

$$\sum_{j \in E(S)} x_j \leq |S| - 1$$

is also a valid inequality.

Example 1 (part a): Suppose the solution to the LP relaxation of a MIP with knapsack constraint

$$4x_1 + 4x_2 + 2x_3 + 2x_4 + 5x_5 + 3x_6 + 9x_7 + 3x_8 + 8x_9 \leq 10$$

is

$$\hat{x}_1 = \frac{1}{2}, \hat{x}_2 = 1, \hat{x}_3 = 1, \hat{x}_4 = 1, \hat{x}_j = 0 \ \forall \ j > 4.$$

The knapsack constraint contains one fractional variable. The solution to (III.5) yields the minimal cover $S = \{1, 2, 3, 4\}$ and the MCC is

$$x_1 + x_2 + x_3 + x_4 \leq 3.$$

Using input $S = \{1, 2, 3, 4\}$ and $N - S = \{5, 6, 7, 8, 9\}$, Procedure 1 computes $\widetilde{S} = \{5, 7, 9\}$ and returns $E(S) = S \cup \widetilde{S} = \{1, 2, 3, 4, 5, 7, 9\}$. The initial MCC has now been lifted to an LMCC:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_7 + x_9 \leq 3.$$

End of Example 1 (part a)

2. Interior Lifting

The linchpin of our facet-finding process is a procedure we call "interior lifting." Interior lifting, unlike simple lifting, affects the composition of the base cover (the minimal cover S found by the XVMCC) because it lifts variables into the LMCC that have constraint

coefficients smaller than $a_{j_{\max}} \equiv \max_{j \in S} a_j$. At the conclusion of interior lifting, the indices corresponding to the $|S|$ smallest constraint coefficients in the revised LMCC form a strong cover S' for the constraint.

Again, we review some of the key definitions. Recall that a strong cover S' is a minimal cover for which either

$$E(S') = N \tag{III.7}$$

or

$$\sum_{j \in S' - \{j_{\max}\}} a_j + a_{i_{\max}} \leq b, \tag{III.8}$$

where

$$j_{\max} = \arg\max_{j \in S'} a_j$$

and

$$a_{i_{\max}} = \max_{i \in N - E(S')} a_i.$$

Given N, coefficients $a_j, \forall\, j \in N$, a minimal cover S, and $E(S)$, the following interior lifting procedure creates a strong cover cut from a lifted minimal cover cut.

PROCEDURE 2 (Interior lifting algorithm): This algorithm takes N, $a_j, \forall\, j \in N$, S, and $E(S)$ and returns the extension of a strong cover $E(S')$.

Input: N, $a_j, \forall\, j \in N$, S, and $E(S)$.

Output: $E(S')$, the extension of a strong cover S'

Begin

$S_0 = S$.

$E(S_0) = E(S)$.

$\widetilde{S_0} = E(S_0) - S_0$.

$g = 0$

While $g \leq |N - E(S_0)| - 1$

$j_{\max} = \arg\max_{j \in S_g} a_j$

$a_{i_g} = \max_{j \in N - E(S_g)} a_j$

If $a_{i_g} \leq b - \sum_{j \in S_g - \{j_{\max}\}} a_j$

Go to **TERMINATE**

Else

$S_{g+1} = S_g + \{i_g\} - \{j_{\max}\}$
$\widetilde{S}_{g+1} = \widetilde{S}_g + \{j_{\max}\}$
$g = g + 1$
Endif
Endwhile
TERMINATE: Return $E(S') \equiv S_g \cup \widetilde{S}_g$.
End

If an $a_{i_g}, i_g \in N - E(S)$ is found such that $a_{i_g} \leq b - \sum_{j \in S_g - \{j_{\max}\}} a_j$, condition (III.8) is met. If all $a_{i_g} > b - \sum_{j \in S_g - \{j_{\max}\}} a_j$, $g = 0, 1, ..., |N - E(S_0)| - 1$, then condition (III.7) is met. In either case, the resulting cover is a strong cover, and the variables corresponding to a lifted strong cover valid inequality enable the formation of a strong cover cut

$$\sum_{j \in E(S')} x_j \leq |S'| - 1, \tag{III.9}$$

where S' is a strong cover. Note that

$$\sum_{j \in S'} x_j \leq |S'| - 1$$

is a strong cover valid inequality, but may not be a cut because some or all of the variables with positive LP solution values may have been removed from the original minimal cover S and placed in $\widetilde{S}'$ of the strong cover. However, because all such variables migrated from the original cover into $\widetilde{S}'$, $S \subseteq E(S') = S' \cup \widetilde{S}'$ and since the right-hand side of the valid inequality did not change, (III.9) is also violated.

Theorem 3.1 *Let $E(S)$ be the extension of a minimal cover for (III.1) corresponding to a minimal cover cut. If S is not strong, successive minimal covers can then be created using Procedure 2 that will result in the formation of the extension a strong cover $E(S')$. Furthermore, this extension will correspond to a strong cover cut. (Thus, we start and end with a violated valid inequality.)*

Proof. If $S_0 \equiv S$ is not strong, there is some

$$a_{i_{\max}} < a_{j_{\max}}, \quad i_{\max} \in N - E(S_0)$$

such that

$$\sum_{j \in S_0 - \{j_{\max}\}} a_j + a_{i_{\max}} > b$$

where

$$j_{\max} = \arg\max_{j \in S_0} a_j$$

and

$$a_{i_{\max}} = \max_{i \in N - E(S_0)} a_i.$$

Recall that $E(S_0) = S_0 \cup \widetilde{S}_0$ where

$$\widetilde{S}_0 = \{i \in N - S_0 \mid a_i \geq a_j, \forall j \in S_0\}.$$

Since

$$a_{i_{\max}} > b - \sum_{j \in S_0 - \{j_{\max}\}} a_j,$$

a new minimal cover S_1 is created

$$S_1 = S_0 - \{j_{\max}\} + \{i_{\max}\}.$$

A new extension is created

$$\widetilde{S}_1 = \widetilde{S}_0 \cup \{j_{\max}\}$$

which is valid because

$$a_{j_{\max}} \geq a_i, \ \forall \ i \in S_1.$$

Let S_g be the gth minimal cover obtained by applying this procedure iteratively. If

$$a_{i_g} > b - \sum_{j \in S_g - \{j_{\max}\}} a_j$$

where

$$j_{\max} = \arg\max_{j \in S_g} a_j$$

and

$$a_{i_g} = \max_{i \in N - E(S_g)} a_i,$$

let

$$S_{g+1} = S_g - \{j_{\max}\} + \{i_g\}\,,$$

and

$$\widetilde{S}_{g+1} = \widetilde{S}_g \cup \{j_{\max}\}\,.$$

The procedure may terminate before examining all of the $|N - E(S_0)|$ coefficients. Suppose at iteration k

$$a_{i_k} \leq b - \sum_{j \in S_k - \{j_{\max}\}} a_j,$$

where

$$a_{i_k} = \max_{i \in N - E(S_k)} a_i.$$

Then,

$$\sum_{j \in S_k - \{j_{\max}\}} a_j + a_i \leq b \;\; \forall i \in N - E(S_k),$$

condition (III.8) is met, and $S' \equiv S_k$ is therfore a strong cover. If all $|N - E(S_0)| = p$ coefficients are lifted in, $E(S_p) = N$, condition (III.7) is met, and $S' \equiv S_p$ is a strong cover. Furthermore, since the index of any variable that leaves the initial minimal cover must belong to the extension of the strong cover, the lifted strong cover valid inequality must be violated just as the minimal cover cut is, i.e., the resulting lifted strong cover valid inequality is a strong cover cut for $\widehat{\mathrm{x}}$. *QED*

Example 1 (part b) Continuing with Example 1, we now must consider variables in $N - E(S)$ (identified below) with coefficients less than $a_{j_{\max}}$, where $a_{j_{\max}} = \max_{j \in S} a_j$. Let $S = S_0$. We are considering the constraint

$$4x_1 + 4x_2 + 2x_3 + 2x_4 + 5x_5 + 3x_6 + 9x_7 + 3x_8 + 8x_9 \leq 10$$

and the cut

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_7 + x_9 \leq 3,$$

where $S_0 = \{1,2,3,4\}$, $\widetilde{S}_0 = \{5,7,9\}$, $E(S_0) = S_0 \cup \widetilde{S}_0 = \{1,2,3,4,5,7,9\}$, $N - E(S_0) = \{6,8\}$, and $a_{j_{\max}} = a_1 = 4$. (Note: $a_{j_{\max}} = a_2$ would also be correct.) Since $\sum_{j \in S - \{j_{\max}\}} a_j + a_6 > b$, condition (III.8) is not met, and S_0 is not strong. Applying Procedure 2, S_1 is now formed as

$$S_1 = S_0 - \{1\} + \{6\}$$

and $\widetilde{S}_1$ is formed as

$$\widetilde{S}_1 = \widetilde{S}_0 + \{1\}.$$

Now we have $S_1 = \{2,3,4,6\}$, $\widetilde{S}_1 = \{1,5,7,9\}$, $E(S_1) = S_1 \cup \widetilde{S}_1 = \{1,2,3,4,5,6,7,9\}$ and $N - E(S_1) = \{8\}$. Since

$$\sum_{j \in S - \{j_{\max}\}} a_j + a_8 \leq b,$$

S_1 is strong, so $S' \equiv S_1$. The strong cover cut is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_9 \leq 3. \qquad \text{(III.10)}$$

Recall that the initial cut began with variables positive in the solution to the LP relaxation. They were

$$\hat{x}_1 = \frac{1}{2}, \hat{x}_2 = 1, \hat{x}_3 = 1, \hat{x}_4 = 1.$$

Since all variables from the initial cut are in this lifted strong cover valid inequality and the right hand side of the cut has remained the same, (III.10) is a cut.

Note that since $a_6 = a_8$, we can exchange x_6 with x_8 in (III.10). This new valid inequality is still violated, and would lead to a different knapsack facet than the facet we find in the next section's example.

End Example 1 (part b)

C. TIGHTENING A STRONG COVER CUT TO A FACET

A strong cover cut that is not a facet can be tightened to a violated knapsack facet using techniques we develop here. We call the strongest of covers a "tight" cover. A strong cover is tight if $\sum_{j \in S'} a_j - b = 1$. For strong cover cuts with tight covers, the facet coefficients

are $\omega_i = 1$ for all $i \in S'$, and are determined by a simple rule based on the value of a_i for all $i \in \widetilde{S}' = E(S') - S'$. However, for strong cover cuts that are not tight ($\sum_{j \in S'} a_j - b > 1$), determining the facet coefficients ω_i for $i \in \widetilde{S}'$ requires a more complex procedure. The algorithm developed by Zemel (1989) allows us to compute the facet coefficients for a minimal cover, and we adapt it for use on strong covers that are not tight, allowing us to make some simplifications. The recursive procedure we develop can be viewed as a simple extension of interior lifting that examines the constraint coefficients a_i of the variables with indices in $\widetilde{S}'$ and assigns to each variable the appropriate facet coefficient ω_i. Thus, we have two tightening results based on the "tightness" of the strong cover, i.e., on the value of $\sum_{j \in S'} a_j - b$, that tighten strong cover cuts to violated facets of the knapsack polytope.

1. Tightening Result I

We would like to be able to directly assign the appropriate facet coefficients based on the value of a_j, $j \in \widetilde{S}'$. However, this can only be done when the constraint coefficients a_j meet certain conditions. We state, without proof, a theorem by Balas:

Theorem 3.2 *(Balas, 1975) Let $S = \{j_1, j_2, ..., j_{|S|}\}$ be a minimal cover for (III.1) ordered so that $a_{j_1} \leq a_{j_2} \leq ... \leq a_{j_{|S|}}$. Let $E(S)$ be the extension of S, and $S_h = \{j_{|S|-h+1}, j_{|S|-h+2}, ..., j_{|S|}\}$; i.e., S_h corresponds to the index set of the h largest coefficients in the ordering of S. Let N be partitioned into $N_0, N_1, ..., N_q$, $q = |S| - 1$, where*

$$N_0 = N - E(S),$$

$$N_1 = S \cup \{i \in E(S) - S | a_i < \sum_{j \in S_2} a_j\},$$

$$N_h = \left\{ i \in E(S) \,|\, \sum_{j \in S_h} a_j \leq a_i < \sum_{j \in S_{h+1}} a_j \right\}, \quad h = 2, ..., q,$$

and define

$$\pi_j = h \;\forall\; j \in N_h,\; h = 0, 1, ..., q.$$

Then, the inequality

$$\sum_{j \in S} x_j + \sum_{j \in N - S} \pi_j x_j \leq |S| - 1 \tag{III.11}$$

is satisfied by all $\mathbf{x} \in P$. Furthermore, if

$$a_i \leq b - \sum_{j \in S - S_{h+1}} a_j,\; \forall\, i \in N_h,\; h = 0, 1, ..., q, \tag{III.12}$$

then (III.11) is a facet of P.

Remark. Condition (III.12) implies that S is a strong cover.

Note that Balas' condition (III.12) really imposes a much stronger condition than a strong cover. The only requirement for a strong cover is

$$\sum_{j \in S' - \{j_{\max}\}} a_j + a_{i_{\max}} \leq b$$

where

$$j_{\max} = \arg\max_{j \in S'} a_j$$

and

$$a_{i_{\max}} = \max_{i \in N - E(S')} a_i.$$

This is equivalent to $a_i \leq b - \sum_{j \in S - S_1} a_j, \forall\, i \in N_0$, which is only one of the $|S|$ conditions of (III.12).

Example 2: Given the knapsack constraint

$$5x_1 + 5x_2 + 6x_3 + 6x_4 + 7x_5 + 8x_6 + 9x_7 + 10x_8 \leq 11$$

and a minimal cover $S = \{1, 2, 3\}$, it is obvious that $E(S) = N$, and the cover is strong. Balas' theorem places the entire extension in $N_1 = \{4, 5, 6, 7, 8\}$ yet only the coefficient of x_4 meets condition (III.12). Thus, the valid inequality derived by the theorem,

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 2,$$

is not a facet. The facet, which can be derived by Zemel's algorithm, is

$$x_1 + x_2 + x_3 + x_4 + 2x_5 + 2x_6 + 2x_7 + 2x_8 \leq 2.$$

End Example 2

Because our goal is to derive facets, we state the following corollary to Balas' theorem, which shows that if S' is a tight cover (i.e., $\sum_{j \in S'} a_j - b = 1$) Balas' theorem is guaranteed to derive a facet.

Corollary 3.3 *If S' is a tight cover, then the inequality (III.11) created by Balas' theorem*

is a facet.

Proof. If

$$\sum_{j \in S'} a_j - b = 1,$$

then

$$\sum_{j \in S' - S'_{h+1}} a_j + \sum_{j \in S'_{h+1}} a_j - b = 1$$

or

$$\sum_{j \in S'_{h+1}} a_j - 1 = b - \sum_{j \in S' - S'_{h+1}} a_j. \tag{III.13}$$

Balas' theorem assigns $\pi_i = h$ for all i such that

$$\sum_{j \in S'_h} a_j \leq a_i < \sum_{j \in S'_{h+1}} a_j, \forall i \in N_h,\ h = 0, 1, ..., q. \tag{III.14}$$

Because of the integrality requirements for b and all a_j, (III.14) can be rewritten as

$$\sum_{j \in S'_h} a_j \leq a_i \leq \sum_{j \in S'_{h+1}} a_j - 1, \forall i \in N_h,\ h = 0, 1, ..., q,$$

which, substituting for the right-hand side using (III.13) yields

$$\sum_{j \in S'_h} a_j \leq a_i \leq b - \sum_{j \in S' - S'_{h+1}} a_j, \forall i \in N_h,\ h = 0, 1, ..., q,$$

which are the conditions specified in (III.12). Thus, when $\sum_{j \in S'} a_j - b = 1$, the coefficients assigned by Balas' theorem must meet conditions (III.12), and the derived valid inequality (III.11) defines a facet of P. *QED*

Note that when the cover is tight, (III.13) allows (III.14) to be rewritten as

$$b - \sum_{j \in S' - S'_h} a_j < a_i \leq b - \sum_{j \in S' - S'_{h+1}} a_j, \forall i \in N_h, h = 0, 1, ..., q.$$

We will use these intervals in the following procedure to assign facet coefficients, but we redefine $\sum_{j \in S' - S'_h} a_j$ as the partial sum $A(s - h)$, the sum of the $s - h$ smallest a_j, $j \in S'$ where $s = |S'|$.

The following procedure applies Corollary 3.3.

Given a tight strong cover S', and the partial sums $A(z)$, $z = 1, ..., s$, defined as the sum of the t smallest a_j, $j \in S'$, the following algorithm assigns facet coefficients for all variables x_j, $j \in N$.

PROCEDURE 3 (Tightening algorithm I): This algorithm takes a tight strong cover S' and the constraint coefficients in $\widetilde{S}' = E(S') - S'$ and returns the facet coefficients ω_i $\forall$ $i \in \widetilde{S}'$.

Input: $\widetilde{S}' = E(S') - S'$ and S'.

Output: ω_i for all $i \in \widetilde{S}'$.

Begin
  Order $i \in \widetilde{S}'$ such that $a_{i_1} \leq a_{i_2} \leq ... \leq a_{i_{|\widetilde{S}'|}}$.
  Order $i \in S'$ such that $a_{j_1} \leq a_{j_2} \leq ... \leq a_{j_{|S'|}}$.
  $A(0) = 0$.
  Compute $A(z)$, $z = 1, 2, ..., |S'|$.
  $s = |S'|$.
  $k = 1$
  $h = 1$
  For $k = 1, 2, ..., \left|\widetilde{S}'\right|$
    If $a_{i_k} \leq b - A(s - (h + 1))$
      $\omega_i = h$
      Increment k
    Else
      Increment h
    Endif
  Endfor
End

The complexity of this algorithm is $O(|S'| \log |S'|) + O(\left|\widetilde{S}'\right| \log \left|\widetilde{S}'\right|) + O(|S'| + \left|\widetilde{S}'\right|)$, which is no worse than $O(n \log n)$.

Example 1 (part c) Finishing the problem begun in the first example, we first check $\sum_{j \in S'} a_j - b$. Since $\sum_{j \in S'} a_j - b = 1$, Corollary 3.3 applies, and we assign ω_i using Procedure 3. Recall that the constraint is

$$4x_1 + 4x_2 + 2x_3 + 2x_4 + 5x_5 + 3x_6 + 9x_7 + 3x_8 + 8x_9 \leq 10,$$

the tight cover is $S' = \{2, 3, 4, 6\}$, and the strong cover cut is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_9 \leq 3,$$

with $E(S') = \{1, 2, 3, 4, 5, 6, 7, 9\}$. We first calculate the set of limits, and the sets defined on these limits to assign the appropriate coefficient ω_j for variable x_j:

$$N_h = \{i \in E(S') \mid b - A(s - h) < a_i \leq b - A(s - (h + 1))\}, \quad h \in \{2, 3\}.$$

Any $j \in E(S')$ such that $6 < a_j \leq 8$ has $\omega_j = 2$, so $\omega_9 = 2$ and any $j \in E(S')$ such that $8 < a_j \leq 10$ has $\omega_j = 3$, so $\omega_7 = 3$. To determine members of $E(S')$ that are assigned the facet coefficient $\omega_j = 1$, we use

$$N_1 = S' \cup \{i \in E(S') - S' \mid a_i \leq b - A(s - 2)\}.$$

Variables $x_j,\ j \in \{1, 2, 3, 4, 5, 6\}$ meet this criterion. The cut is now

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 3x_7 + 2x_9 \leq 3.$$

This valid inequality must be a facet of the polytope of the original knapsack constraint because it satisfies the condition in the corollary. Recall again that we began this tightening process with a valid inequality which was a strong cover cut. The variables positive in the solution to the LP relaxation were

$$\hat{x}_1 = \frac{1}{2}, \hat{x}_2 = 1, \hat{x}_3 = 1, \hat{x}_4 = 1.$$

Since all variables from the strong cover cut are in this facet with facet coefficients $\omega_j \geq 1$, and the right hand side of the valid inequality is the same as that of the strong cover cut, this facet is also a cut because

$$\sum_{j \in S'} \widehat{x}_j + \sum_{j \in \widetilde{S}'} \omega_j \widehat{x}_j > |S'| - 1.$$

End of Example 1 (part c)

2. Tightening Result II

If the strong cover is not tight, i.e., if

$$\sum_{j \in S'} a_j - b > 1,$$

we cannot assign facet coefficients using Corollary 3.3. The recursive procedure we develop next, which is similar to Zemel (1989) assigns the appropriate facet coefficients in polynomial time. Zemel developed his algorithm to solve Padberg's series of problems that assign facet coefficients. First, we examine Padberg's result.

a. Padberg's result

A facet-finding result by Padberg (1975) has been the foundation for many of the critical discoveries of the knapsack polytope. This result, stated without proof, creates a facet of the knapsack polytope by sequentially lifting variables into an MCVI.

Theorem 3.4 *(Padberg 1975). Let S be a minimal cover for (III.1), let $N-S = \{i_1, ..., i_p\}$ be arbitrarily ordered, and consider the sequence of knapsack problems $KS(i_k)$ defined recursively as*

$$\begin{aligned} KS(i_k) \qquad z_{i_k} \;=\; & \textit{maximize} \quad \sum_{j \in S} x_j + \sum_{i=i_1}^{i_{k-1}} \beta_i x_i \\ & \textit{subject to:} \quad \sum_{j \in S} a_j x_j + \sum_{i=i_1}^{i_{k-1}} a_i x_i \;\leq\; b - a_{i_k} \\ & \qquad\qquad x_j \;\in\; \{0,1\} \;\forall\, j \in S \\ & \qquad\qquad x_i \;\in\; \{0,1\} \;\forall\, i \in N - S \end{aligned}$$

for $k = 1, ..., p$ (where summation over the empty set is 0), with the coefficients β_i defined by

$$\beta_i = |S| - 1 - z_i, \;\; i = i_1, ..., i_{k-1}.$$

Then, the inequality

$$\sum_{j \in S} x_j + \sum_{i \in N-S} \beta_i x_i \leq |S| - 1$$

is a facet of P.

The theorem implies that a series of $|N - S|$ IPs (knapsack problems) must be solved to determine the coefficients β_i.

b. Zemel's result

Zemel recognized that Padberg's problem could be solved in a much simpler way. First, he recognized that Padberg's procedure was related to the set of dual knapsack problems $DKS_i(z)$, $z = 0, ..., s-1$ where $s = |S|$.

$$
\begin{array}{lllll}
DKS_{i_k}(z) \quad A_{i_k}(z) = \text{minimize} & \sum\limits_{j \in S} a_j x_j + \sum\limits_{i=i_1}^{i_{k-1}} a_i x_i & & & \\
\text{subject to:} & \sum\limits_{j \in S} x_j + \sum\limits_{i=i_1}^{i_{k-1}} \beta_i x_i & \geq & z & \\
& x_j & \in & \{0,1\} \; \forall \, j \in S & \\
& x_i & \in & \{0,1\} \; \forall \, i \in N - S &
\end{array}
$$

The problem $KS(i_k)$ is related to the set of problems $DKS_{i_k}(z)$, $z = 0, ..., s-1$ via the relation $z_{i_k} = \max \{z : A_{i_k}(z) \leq b - a_{i_k}\}$.

ALGORITHM LIFT (Zemel, 1989): This algorithm assigns facet coefficients for all variables x_j, $j \in N$ in polynomial time. This algorithm takes a minimal cover S, the partial sums ℓ_t, $t = 1, ..., s-1$, defined as follows:

$$
\begin{aligned}
\ell_1 &= a_{j_1}, \text{ where } j_1 = \operatorname*{argmin}_{j \in S} a_j, \\
\ell_2 &= \ell_1 + a_{j_2}, \text{ where } j_2 = \operatorname*{argmin}_{j \in S - \{j_1\}} a_j, \\
&\vdots \\
\ell_{s-1} &= \ell_{s-2} + a_{j_{s-1}}, \text{ where } j_{s-1} = \operatorname*{argmin}_{j \in S - \{j_1\} - \ldots - \{j_{s-2}\}} a,
\end{aligned}
$$

the right-hand side b and the constraint coefficients in $\widetilde{S} = E(S) - S$ and returns the facet coefficients $\beta_i \; \forall \, i \in \widetilde{S}$.

Input: $\widetilde{S} = E(S) - S$ (in any sequence) for a minimal cover S, the right-hand side b, $a_i \; \forall \, i \in \widetilde{S}$, and the partial sums ℓ_t, $t = 1, ..., s-1$.

Output: β_i for all $i \in \widetilde{S}$.

Begin
Let $A_1(0) = 0, A_1(z) = \ell_z$, $z = 1, ..., s-1$

For $i = 1, ..., \left|\widetilde{S}\right|$
$z_i = \max\{z : A_1(z) \leq b - a_i\}$
$\beta_i = s - 1 - z_i$
For $z = 0, ..., s-1$
If $z < \beta_i$, $A_{i+1}(z) = A_i(z)$
Else $A_{i+1}(z) = \min\left\{A_i(z), A_i(z - \beta_{\pi_i}) + a_i\right\}$
Endif
Endfor
Endfor
End

The facet that corresponds to this algorithm is

$$\sum_{j \in S} x_j + \sum_{i \in \widetilde{S}} \beta_i x_i \leq |S| - 1.$$

Note that the facet produced will depend on the order in which the elements of $\widetilde{S}$ are processed.

In practice, we will apply a variant of Zemel's algorithm after we have lifted to a strong cover cut and determined that the strong cover cut is not tight. Because we begin the process with a strong cover cut, the facet we find will be a violated facet.

c. Tightening Algorithm II

We will modify Zemel's algorithm to efficiently tighten strong covers. First, by calculating (and updating) one additional sum, we may be able to terminate the algorithm after only assigning a few facet coefficients, leaving the others to be assigned by Procedure 3, a much simpler procedure. The use of this additional sum will also allow us to avoid many of the checks $\min\{A_i(z), A_i(z - \beta_i) + a_i\}$ made in Algorithm Lift.

Let $s = |S'|$ and define $A_i(s) = \sum_{j \in S'} a_j$.

Corollary 3.5 *If $A_i(s) - b = 1$, then all remaining $\left|\widetilde{S'}\right| - i$ facet coefficients can be assigned using Procedure 3 with the current partial sums $A_i(z)$, $z = 0, 1, ..., s$.*

Proof. If $A_i(s) - b = 1$, we will show that

$$A_i(z - \beta_i) + a_i \geq A_i(z), \; z = 0, 1, ..., s.$$

Therefore, the assignment

$$A_{i+1}(z) = \min \{A_i(z), A_i(z - \beta_i) + a_i\}$$

becomes

$$A_{i+1}(z) = A_i(z),$$

and

$$A_{i+k}(z) = A_i(z),\ k = 2, ..., \left|\widetilde{S}'\right|,$$

which allows coefficients

$$\beta_{i+k},\ k = 1, ..., \left|\widetilde{S}'\right|$$

to be assigned using the partial sums

$$A_i(z),\ z = 0, 1, ..., s.$$

Suppose $A_i(s) - b = 1$, but $A_i(z - \beta_i) + a_i < A_i(z)$ for some $z = \beta_i, \beta_i + 1, ..., s$. We know that x_i is assigned coefficient β_i when

$$A_i(s - 1 - \beta_i) + a_i \leq b \text{ and } A_i(s - \beta_i) + a_i > b.$$

Since

$$A_i(s - \beta_i) + a_i > b,$$

and

$$A_i(s) - 1 = b,$$

then

$$A_i(s - \beta_i) + a_i > A_i(s) - 1,$$

and, because of the integrality of the data,

$$A_i(s - \beta_i) + a_i \geq A_i(s).$$

Because of the way the partial sums were formed and updated, we know that

$$A_i(s) - A_i(s - \beta_i) \geq A_i(s - k) - A_i(s - \beta_i - k),\ k = 1, 2, ..., s - \beta_i,$$

and since

$$a_i \geq A_i(s) - A_i(s - \beta_i),$$

we know that

$$a_i \geq A_i(s - k) - A_i(s - \beta_i - k),\ k = 1, 2, ...s - \beta_i.$$

Equivalently

$$A_i(s - \beta_i - k) + a_i \geq A_i(s - k),\ k = 1, 2, ...s - \beta_i$$

which is a contradiction. There cannot exist $A_i(z - \beta_i) + a_i < A_i(z)$ for any $z = 1, ..., s$. So,

$$A_{|\tilde{S}|}(z) = A_{|\tilde{S}|-1}(z) = ... = A_{i+1}(z) = A_i(z),\ z = 0, 1, ..., s,$$

and all unassigned facet coefficients can be assigned using Procedure 3 with the current set of partial sums $A_i(z)$, $z = 0, 1, ..., s$.

QED

Corollary 3.6 *If $a_i \geq A_i(s) - A_i(s - \beta_i)$, all partial sums used in iteration $i+1$ will be the same as those used in iteration i of Algorithm Lift, that is, $A_{i+1}(z) = A_i(z)$, $z = 0, 1, ..., s$.*

Proof. If $a_i \geq A_i(s) - A_i(s - \beta_i)$, we will show that

$$A_i(z - \beta_i) + a_i \geq A_i(z),\ z = 0, 1, ..., s.$$

Therefore, the assignment

$$A_{i+1}(z) = \min \{A_i(z), A_i(z - \beta_i) + a_i\}$$

becomes

$$A_{i+1}(z) = A_i(z),\ z = \beta_i, \beta_i + 1, ..., s.$$

Suppose $a_i \geq A_i(s) - A_i(s - \beta_i)$, but $A_i(z - \beta_i) + a_i < A_i(z)$. Because of the way the partial sums were formed and updated, we know that

$$A_i(s) - A_i(s - \beta_i) \geq A_i(s - k) - A_i(s - \beta_i - k),\ k = 1, 2, ..., s - \beta_i$$

and since

$$a_i \geq A_i(s) - A_i(s - \beta_i),$$

we know that

$$a_i \geq A_i(s-k) - A_i(s-\beta_i-k),\ k = 1, 2, ...s-\beta_i.$$

Equivalently

$$A_i(s-\beta_i-k) + a_i \geq A_i(s-k),\ k = 1, 2, ...s-\beta_i$$

which is a contradiction. There cannot exist $A_i(z-\beta_i) + a_i < A_i(z)$ for any $z = 1, ..., s$. So,

$$A_{i+1}(z) = A_i(z),\ z = 0, 1, ..., s,$$

and all partial sums used in iteration $i+1$ will be the same as those used in iteration i.

QED

Corollary 3.7 *If the x_i, $i \in \widetilde{S}$ are assigned facet coefficients in ascending order of the constraint coefficients a_i, $i \in \widetilde{S}$, the algorithm can be terminated when the first x_i receives facet coefficient $\beta_i = \left\lfloor \frac{s}{2} \right\rfloor + 1$, and the remaining x_i, $i \in \widetilde{S}$ can be directly assigned facet coefficients using Procedure 3 with the current partial sums $A_i(z)$, $z = 0, 1, ..., s-1-\beta_i$.*

Proof. When $\beta_i = \left\lfloor \frac{s}{2} \right\rfloor + 1$,

$$A_i(s-1-\left(\left\lfloor \frac{s}{2} \right\rfloor + 1\right)) \leq b - a_i.$$

The smallest partial sum that can be adjusted is

$$A_{i+1}(\beta_i) = \min\left\{A_i(\beta_i), A_i(z-\beta_i) + a_i\right\}.$$

Because of the imposed ordering, $a_{i+1} \geq a_i$. This means that $b - a_{i+1} \leq b - a_i$, and thus, the largest partial sum smaller than the difference $b - a_{i+1}$ can be no larger than

$$A_{i+1}(s-1-\left(\left\lfloor \frac{s}{2} \right\rfloor + 1\right)) = A_{i+1}(s-\left\lfloor \frac{s}{2} \right\rfloor - 2).$$

Since

$$A_{i+1}(s-\left\lfloor \frac{s}{2} \right\rfloor - 2) < A_{i+1}(\left\lfloor \frac{s}{2} \right\rfloor + 1) = A_{i+1}(\beta_i),$$

the remaining $i \in \widetilde{S}$ will be assigned facet coefficients based only on the partial sums that will never be adjusted if the procedure were allowed to continue, that is

$$A_{|\widetilde{S}|}(z) = A_{|\widetilde{S}|-1}(z) = ... = A_{i+1}(z) = A_i(z), \forall z < \beta_i.$$

QED

Given the following corollaries, we modify Algorithm Lift to assign facet coefficients. We begin with a strong cover S' that is not tight, and define the partial sum ℓ_t, $t = 1, ..., s$, as the sum of the t smallest a_j, $j \in S'$ where $s = |S'|$.

PROCEDURE 4 (Tightening algorithm II): This algorithm takes the constraint coefficients in $\widetilde{S}' = E(S') - S'$ for the partial sums ℓ_t, $t = 1, ..., s$ for a strong cover S' and returns the cut coefficients ω_i.

Input: $\widetilde{S}' = E(S') - S'$ and the partial sums ℓ_t, $t = 1, ..., s$ for a strong cover S'.

Output: ω_i for all $i \in \widetilde{S}'$.

Begin
Let $A_1(0) = 0, A_1(z) = \ell_z$, $z = 1, ..., s$, and order $\widetilde{S}'$ in ascending order of a_j,
$h = 2$
For $i = 1, 2, ..., \left|\widetilde{S}'\right|$
While $h < \left\lfloor \frac{|S'|}{2} \right\rfloor + 1$
$\underline{I} = b - A_1(s - h)$
$\overline{I} = b - A_1(s - (h + 1))$
If $a_i \leq \underline{I}$, then
$\omega_i = h - 1$
Else
If $a_i \leq \overline{I}$, then
$\omega_i = h$
If $a_i < A_i(s) - A_i(s - h)$ then
If $z < h$, $A_{i+1}(z) = A_i(z)$.
Else $A_{i+1}(z) = \min\{A_i(z), A_i(z - h) + a_i\}$, $z = h, h + 1, ..., s$.
Endif
$i = i + 1$
If $A_i(s) = b + 1$
Terminate algorithm and assign remaining
coefficients using **Procedure 3** with current partial
sums $A_i(z)$, $z = 0, 1..., s$.
Endif
Endif
Else
$h = h + 1$
Endif
Endif
Endwhile
Use **Procedure 3** with current partial sums $A_i(z)$, $z = 0, 1, ..., s$ to assign
remaining coefficients

Endfor
Return ω_i for each $i \in \widetilde{S}'$
End

Once all ω_i have been assigned, the strong cover cut has been lifted to

$$\sum_{j \in S'} x_j + \sum_{i \in \widetilde{S}'} \omega_i x_i \leq |S'| - 1$$

which is a violated facet of the knapsack polytope.

We illustrate Procedure 4 with a simple example:

Example 3: Given the knapsack constraint

$$4x_1 + 5x_2 + 5x_3 + 6x_4 + 7x_5 + 8x_6 + 8x_7 + 12x_8 + 13x_9 \leq 16,$$

and a strong cover is $S' = \{1, 2, 3, 4\}$, $s = 4$, the corresponding strong cover cut is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3,$$

and the initial partial sums are

$$A_1(0) = 0,\ A_1(1) = 4,\ A_1(2) = 9,\ A_1(3) = 14,\ A_1(4) = 20.$$

Because the smallest a_i that can get a facet coefficient must be greater than $b - A_1(s-2) = 7$, $\omega_5 = 1$.

Since

$$b - A_1(s-2) < a_6 \leq b - A_1(s-3),$$

$\omega_6 = 2$, and since

$$a_6 < A_1(s) - A_1(s-2) = 11,$$

we must adjust the partial sums $A_2(z)$:

For $z < 2$,

$$A_2(0) = A_1(0) = 0,\ A_2(1) = A_1(1) = 4$$

For $z = 2, 3, ..., s$, $A_2(z) = \min\{A_1(z), A_1(z-2) + a_6\}$. Thus,

$$\begin{aligned} A_2(2) &= \min\{9, 0+8\} = 8, \\ A_2(3) &= \min\{14, 4+8\} = 12, \end{aligned}$$

$$A_2(4) = \min\{20, 9+8\} = 17.$$

Since $A_2(s) = A_2(4) = 17 = b+1$, we can assign the remaining coefficients using Procedure 3 with the partial sums $A_2(z)$, $z = 0, 1, ..., s$. The facet is

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 + x_7 + 2x_8 + 3x_9 \leq 3.$$

End Example 3

Next, we illustrate Procedure 4 with a more complex example:

Example 4: Given the knapsack constraint

$$40x_1 + 41x_2 + 42x_3 + 43x_4 + 44x_5 + 75x_6 + 115x_7 + 140x_8 \leq 170,$$

the strong cover is $S' = \{1, 2, 3, 4, 5\}$, and the strong cover cut is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 4,$$

and the initial partial sums are

$$A_1(0) = 0,\ A_1(1) = 40,\ A_1(2) = 81,\ A_1(3) = 123,\ A_1(4) = 166,\ A_1(5) = 210.$$

The first variable we consider is x_6, the variable with the smallest a_i in $\widetilde{S}'$. Because

$$b - A_1(s-2) < a_6 \leq b - A_1(s-3),$$

$\omega_6 = 2$, and since

$$a_6 < A_i(s) - A_i(s-2) = 87,$$

we must adjust the partial sums $A_2(z)$:

For $z < 2$,

$$A_2(0) = A_1(0) = 0,\ A_2(1) = A_1(1) = 40.$$

For $z = 2, 3, ..., s$, $A_2(z) = \min\{A_1(z), A_1(z-2) + a_6\}$. Thus,

$$A_2(2) = \min\{81, 0+75\} = 75$$

$$A_2(3) = \min\{123, 40+75\} = 115$$

$$A_2(4) = \min\{166, 81+75\} = 156$$
$$A_2(5) = \min\{210, 123+75\} = 198.$$

Since $A_2(s) = A_2(5) = 198 > b+1$, we cannot revert to Procedure 3.

We consider x_7. Because

$$b - A_2(s-3) < a_7 \leq b - A_2(s-4),$$

$\omega_7 = 3$. We can revert to Procedure 3 according to Corollary 3.7 because $\omega_7 = 3 = \lfloor \frac{s}{2} \rfloor + 1$. Instead, we will continue the process to verify the correctness of Corollary 3.7.

Since

$$a_7 < A_2(s) - A_2(s-3) = 123,$$

we adjust the partial sums again.

For $z < 3$,

$$A_3(0) = A_2(0) = 0,\ A_3(1) = A_2(1) = 40,\ A_3(2) = A_2(2) = 75.$$

For $z = 3, ..., s$, $A_3(z) = \min\{A_2(z), A_2(z-3) + a_7\}$. Thus,

$$A_3(3) = \min\{115, 0+115\} = 115$$
$$A_3(4) = \min\{156, 40+115\} = 155$$
$$A_3(5) = \min\{198, 75+115\} = 190.$$

Since

$$A_3(s) = A_3(5) = 190 > b+1$$

we cannot revert to Procedure 3 (via Corollary 3.5).

We consider x_8. Because

$$b - A_3(s-4) < a_8 \leq b - A_3(s-5),$$

$\omega_8 = 4$. Note that the only partial sums we use are

$$A_3(s-4) = A_3(1) = A_2(1)$$

and

$$A_3(s-5) = A_3(0) = A_2(0).$$

In fact, if we change the coefficient a_8 and let it be the smallest possible value given the value of a_7 and the imposed ordering, $a_8 = 115$, we can see that in the worst case,

$$b - A_3(s-3) < a_8 \leq b - A_3(s-4),$$

and the only partial sums that would be used are $A_3(2) = A_2(2)$ and $A_3(1) = A_2(1)$. Thus, the last adjustment was unnecessary, and we could have assigned $\omega_8 = 4$ based on the partial sums $A_2(z)$, $z = 0, 1$. The facet is

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 + 3x_7 + 4x_8 \leq 4.$$

End Example 4

D. NON-MINIMAL COVER CUTS FROM THE KNAPSACK POLYTOPE

Although a knapsack constraint may contain one or more variables with LP solution values $\widehat{x}_j$ that are fractional, there is no guarantee that an MCC exists. In the event no minimal cover cut exists for a candidate knapsack constraint, we develop a method that finds a "non-minimal cover cut" (NMCC), if such a cut exists. We distinguish between two types of NMCCs. A Type I NMCC only requires cut coefficients of 1 for variables in the NMCC, and, if the NMCC is not a violated "extended minimal cover valid inequality" (TMCVI), it is lifted by "deficit lifting" (and possibly simple lifting) to a violated TMCVI that is, in turn, lifted and tightened to a knapsack facet. Type II NMCCs are valid inequalities that require that one or more cut coefficients be greater than 1 to obtain a violated valid inequality.

1. Non-Minimal Cover Cut-Finding Problem

Once a candidate knapsack constraint is identified, the minimal cover cut-finding

algorithm is invoked. In the event that the algorithm finds no minimal cover cut, the NMCC-finding algorithm is called. We define a two-step procedure that first calculates the right-hand side of a "non-minimal cover valid inequality" (NMCVI), and then checks to see if the NMCVI is violated before any lifting or tightening is carried out. Given $\widehat{\mathrm{x}}$, the right-hand side ω_0 is found by solving the following IP (which uses previously defined notation):

Formulation:

$$\begin{aligned} \omega_0^* = \text{ maximize } & \sum_{j \in N^+} x_j & & \\ \text{subject to } & \sum_{j \in N^+} a_j x_j & \leq & \ b \\ & x_j & \in & \ \{0,1\} \ \forall \, j \in N^+. \end{aligned} \tag{III.15}$$

The standard dynamic programming recursion used to solve (III.15) is:

DP Recursion 2

Initial conditions:

$d_0(0) = 0$, $d_0(\ell) = -\infty$ for all $\ell \neq 0$.

Recursion:

$d_k(\ell) = \max\{d_{k-1}(\ell), d_{k-1}(\ell - a_k) + 1\}$ for $k = 1, ..., |N^+|$ and $\ell = 0, ..., b$.

The solution to the problem is $\omega_0^* = \max_{0 \leq \ell \leq b} d_{|N^+|}(\ell)$, and the NMCVI is

$$\sum_{j \in N^+} x_j \leq \omega_0^*. \tag{III.16}$$

The second step of the cut-finding process is simply to check the NMCVI for violation. The NMCVI (III.16) is a Type I NMCC if

$$\sum_{j \in N^+} \widehat{x}_j - \omega_0^* > 0. \tag{III.17}$$

An outline of the two-step NMCC-finding procedure follows.

PROCEDURE 5 (Non-minimal cover cut-finding algorithm): This algorithm takes b, the values of $\widehat{x}_j$ and a_j, $j \in N^+$, and returns ω_0 and the message that $\sum_{j \in N^+} x_j \leq \omega_0$ is a Type I NMCC, or returns the message that no Type I NMCC exists.

Input: The values of $\widehat{x}_j$ and a_j, $j \in N^+$ and the right-hand side of the constraint b.

Output: The message that $\sum_{j \in N^+} x_j \leq \omega_0$ is a Type I NMCC and ω_0, or the

message that no Type I NMCC exists.

Begin
  Find ω_0 by DP recursion 2
  If $\sum_{j \in N^+} \widehat{x}_j - \omega_0 > 0$
    Return "NMCVI is a Type I NMCC" and ω_0
  Else
    Return "NMCVI is not a Type I NMCC"
  Endif
End

Example 2 (part a): Suppose the solution to the LP relaxation of a MIP with knapsack constraint

$$2x_1 + 4x_2 + 8x_3 + 8x_4 + 8x_5 \leq 10 \tag{III.18}$$

is

$$\widehat{x}_1 = 1,\ \widehat{x}_2 = 1/4,\ \widehat{x}_3 = 1/4,\ \widehat{x}_4 = 1/4,\ \widehat{x}_5 = 3/8$$

The knapsack constraint has four fractional variables, so it is eligible to have a cut derived from it. Note that any two of the coefficients a_j, $j \in \{2, 3, 4, 5\}$ will form a minimal cover, so there are $\binom{4}{2}$ MCVIs of two variables with the right-hand side of $|S| - 1 = 1$ associated with (III.18). But, because no sum of any two $\widehat{x}_j$, $j \in \{2, 3, 4, 5\}$ exceeds 1, no minimal cover cut exists.

The first step to deriving a possible NMCC is to find the right hand side of the NMCVI by solving (III.15) with DP recursion 2. The solution is $\omega_0 = 2$, and the valid inequality is

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 2. \tag{III.19}$$

Checking condition (III.17) confirms that (III.19) is a Type I NMCC since

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 + \widehat{x}_4 + \widehat{x}_5 > 2.$$

End of Example 2 (part a)

2. Deficit Lifting

We find a facet from the Type I NMCC as we did from the MCC. If the $\omega_0 + 1$ smallest a_j, $j \in N^+$ form a minimal cover, we designate the indices associated with the

$\omega_0 + 1$ smallest a_j, $j \in N^+$ as S and employ Procedures 1-4 as necessary to find a violated knapsack facet. If not, we must utilize a process called "deficit lifting" to find a TMCVI.

Deficit lifting finds a violated valid inequality that can be lifted to a facet by excluding variables from the Type I NMCC. To begin, we designate the variables with the $\omega_0 + 1$ smallest a_j, $j \in N^+$ as members of the (possibly non-minimal) cover S^0_{cvr}.

PROCEDURE 6 (Deficit lifting algorithm): This algorithm takes S^0_{cvr}, the vector of constraint coefficients a_j, $j \in S^0_{cvr}$, the right-hand side of the constraint b, the right-hand side of the NMCVI ω_0, and returns a minimal cover S^M_{cvr} and ω_M, the right-hand side of the NMCVI corresponding to S^M_{cvr}.

Input: S^0_{cvr}, a_j, $j \in S^0_{cvr}$, ω_0 and b.

Output: A minimal cover S^M_{cvr} for the constraint.

Begin
  $g = 0$
  For $g = 0, 1, ..., |\omega_0|$
    $a_{i_g} = \min_{j \in S^g_{cvr}} a_j$
    If $\sum_{j \in S^g_{cvr} - \{i_g\}} a_j \leq b$
      go to **TERMINATE**.
    Else
      $S^{g+1}_{cvr} = S^g_{cvr} - \{i_g\}$
      $\omega_{g+1} = \omega_g - 1$
      $g = g + 1$
    Endif
  Endfor
  Set $S^M_{cvr} = S^g_{cvr}$ and $\omega_M = \omega_g$
  TERMINATE: Return S^M_{cvr} and ω_M
End

The resulting valid inequality is

$$\sum_{j \in S^M_{cvr}} x_j + \sum_{j \in N^+ - S^0_{cvr}} x_j \leq \omega_0. \qquad \text{(III.20)}$$

Of course, it would be pointless to lift a Type I NMCC with deficit lifting if the resulting valid inequality were not violated. Fortunately, the following result proves that a violated valid inequality always results from deficit lifting a Type I NMCC.

Lemma 3.8 *A valid inequality created from a Type I NMCC by deficit lifting will always be violated.*

Proof. If

$$\sum_{j \in S^0_{cvr}} \widehat{x}_j + \sum_{j \in N^+ - S^0_{cvr}} \widehat{x}_j > \omega_0,$$

then

$$\sum_{j \in S^0_{cvr} - \{i_0\}} \widehat{x}_j + \sum_{j \in N^+ - S^0_{cvr}} \widehat{x}_j > \omega_0 - 1,$$

because

$$\widehat{x}_{i_0} \leq 1.$$

The result for S^M_{cvr} follows by induction on M. *QED*

Note that (III.20) is not necessarily a TMCVI at this point. The actual TMCVI is

$$\sum_{j \in S^M_{cvr}} x_j + \sum_{j \in E(S^M_{cvr}) - S^M_{cvr}} x_j \leq \omega_0.$$

However, since all $a_i \geq a_{j\max}$, $i \in N^+ - S^0_{cvr}$ where $a_{j_{\max}} = \max_{j \in S^M_{cvr}} a_j$, then $N^+ - S^0_{cvr} \subseteq E(S^M_{cvr}) - S^M_{cvr}$, and it is a simple task to use simple lifting (Procedure 1) to lift any remaining variables that belong in the TMCVI. After a TMCVI is completely formed, interior lifting (Procedure 2) and tightening (Procedure 3 or Procedure 4) will create a facet, just as they did with a lifted minimal cover cut.

We continue with example 2 to demonstrate deficit lifting:

Example 2 (part b): We have the knapsack constraint

$$2x_1 + 4x_2 + 8x_3 + 8x_4 + 8x_5 \leq 10, \tag{III.21}$$

with LP solution values

$$\widehat{x}_1 = 1,\ \widehat{x}_2 = 1/4,\ \widehat{x}_3 = 1/4,\ \widehat{x}_4 = 1/4,\ \widehat{x}_5 = 3/8,$$

and the NMCC

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 2.$$

Since $\omega_0 = 2$, we form $S^0_{cvr} = \{1, 2, 3\}$. Since $a_1 = a_{i_0} = \min_{j \in S^0_{cvr}} a_j$, the first iteration of

Procedure 5 finds that

$$\sum_{j \in S^0_{cvr} - \{i_0\}} a_j \not\leq b,$$

so we form $S^1_{cvr} = S^0_{cvr} - \{i_0\} = \{2, 3\}$ and $\omega_1 = \omega_0 - 1 = 1$. Since $a_2 = a_{i_1} = \min_{j \in S^1_{cvr}} a_j$, the next iteration of Procedure 5 finds that

$$\sum_{j \in S^1_{cvr} - \{i_1\}} a_j \leq b,$$

so S^1_{cvr} is a minimal cover, and

$$\sum_{j \in S^1_{cvr}} x_j + \sum_{j \in N^+ - S^0_{cvr}} x_j \leq \omega_1$$

is a cut. The new cut is

$$x_2 + x_3 + x_4 + x_5 \leq 1. \tag{III.22}$$

Since no lifting can be performed on (III.22) and tightening assigns cut coefficients of 1 for all variables not in the cover, (III.22) is a facet of the knapsack polytope associated with (III.21).

End of Example 2 (part b)

3. Type II Non-Minimal Cover Cut-Finding Problem

Type II NMCCs are valid inequalities that require that one or more cut coefficients be greater than 1 to obtain a violated valid inequality. Type II NMCCs can be found by a three-step process. First, we solve the NMCC-finding problem. If the solution to the NMCC-finding problem indicates that no Type I NMCC exists, the NMCVI is lifted and tightened to a facet of the associated knapsack polytope much as we did for Type I NMCCs. We then examine the resulting facet-defining valid inequality. If the facet is violated, then we have found a Type II NMCC. If not, we must first check the strong cover S' used to find the facet. If $\sum_{j \in S'} a_j - b = 1$, then we know that we have assigned the maximum cut coefficient to each variable in the cut, and no Type II NMCC exists. If $\sum_{j \in S'} a_j - b > 1$, there may exist other facets that can be found by exchanging facet coefficients among variables in certain equivalence classes. The equivalence classes are defined by

$$b - A_1(s-h) < a_i \leq b - A_{|\widetilde{S}'|}(s-h),\ h = 2, 3, ..., |S'| - 1. \qquad \text{(III.23)}$$

where $A_{|\widetilde{S}'|}(s-h), h = 2, 3, ..., |S'| - 1$ represent the last partial sums used by Procedure 4 to find the first facet. The equivalence classes identify sets of variables that will have facet coefficients of either $\omega_i = h-1$ or $\omega_i = h$. Note that in Procedure 4, the first coefficient that falls in any equivalence class will have a facet coefficient of $\omega_i = h$ and cause partial sums to be adjusted. If two variables are in the same equivalence class and have different facet coefficients, then the facet coefficients can be exchanged so the variable with the greater $\widehat{x}_i$ has the greater ω_i.

The following procedure finds a Type II NMCC, if one exists.

PROCEDURE 7 (Type II Non-minimal cover cut-finding algorithm): This algorithm takes b, the values of $\widehat{x}_j$ and a_j, $j \in N^+$, and ω_0, and returns the facet coefficients ω'_i for x_i, $i \in \widetilde{S}'$, or returns the message that no Type II NMCC exists.

Input: The values of $\widehat{x}_j$ and a_j, $j \in N^+$, the right-hand side of the constraint b, and the right-hand side of the cut ω_0.

Output: The facet coefficients ω'_i for x_i, $i \in \widetilde{S}'$, or the message that no Type I NMCC exists.

Begin
  Lift the NMCVI to a facet by using Procedures 1-6 as required.
  If $\sum_{j \in S'} \widehat{x}_j + \sum_{i \in \widetilde{S}'} \omega_i \widehat{x}_i - \omega_0 > 0$
    Return "Facet is a Type II NMCC," and set $\omega'_i = \omega_i$ for $i \in \widetilde{S}'$.
  Else
    If $\sum_{j \in S'} a_j - b = 1$
      Return "No Type II NMCC exists"
    Else
      For each equivalence class $Q = 2, 3, ..., |S'| - 1$
        For all $i, k \in Q$
          If $\omega_i > \omega_k$ and $\widehat{x}_i < \widehat{x}_k$, $i, k \in Q$
            $\omega'_k = \omega_i$ and $\omega'_i = \omega_k$
          Else
            $\omega'_i = \omega_i$ for all $i \in Q$
          Endif
        Endfor
      Endfor

If $\sum_{j \in S'} \widehat{x}_j + \sum_{i \in \widetilde{S}'} \omega_i' \widehat{x}_i - \omega_0 > 0$

Return ω_i' for x_i, $i \in \widetilde{S}'$

Else

Return "No Type II NMCC exists"

Endif

Endif

End

Example 3: We have the knapsack constraint

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 3x_5 + 3x_6 + 3x_7 + 5x_8 \leq 6 \tag{III.24}$$

with positive LP solution values

$$\widehat{x}_1 = 1/4,\ \widehat{x}_2 = 1/2,\ \widehat{x}_3 = 1/2,\ \widehat{x}_4 = 1,\ \widehat{x}_7 = 1/2$$

and the NMCVI

$$x_1 + x_2 + x_3 + x_4 + x_7 \leq 3. \tag{III.25}$$

Since

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 + \widehat{x}_4 + \widehat{x}_7 = 2\frac{3}{4},$$

(III.25) is not a cut. If we form a TMCVI and lift and tighten it to a facet, we obtain the following valid inequality:

$$x_1 + x_2 + x_3 + x_4 + 2x_5 + x_6 + x_7 + 3x_8 \leq 3$$

but still

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 + \widehat{x}_4 + 2\widehat{x}_5 + \widehat{x}_6 + \widehat{x}_7 + 3\widehat{x}_8 \leq 3$$

because the only variables that got larger coefficients were 0 in the LP solution. We define the equivalence classes using (III.23):

Equivalence Class	$2 < a_i < 4$	$4 < a_i < 4$
Facet coefficients	$\omega_i = 1$ or 2	$\omega_i = 2$ or 3.

Since variables x_5, x_6, and x_7 are all in the same equivalence class (note the second equivalence class is empty), we find that $\omega_5 > \omega_7$ and $\widehat{x}_5 < \widehat{x}_7$, and we interchange ω_5 and ω_7

so that $\omega'_5 = 1$ and $\omega'_7 = 2$. The new facet is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 2x_7 + 3x_8 \leq 3,$$

and since

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 + \widehat{x}_4 + \widehat{x}_5 + \widehat{x}_6 + 2\widehat{x}_7 + 3\widehat{x}_8 = 3\frac{1}{4} > 3,$$

it is a Type II NMCC.

End of Example 3

The procedure to form Type II NMCCs requires that we have a minimal cover from which we generate a facet. However, sequential lifting of facets associated with minimal covers does not, in general, produce all facets associated with the polytope P (Balas and Zemel, 1984). For facets not associated with minimal covers, a procedure similar to our XVMCC cut-finding procedure may be able to produce these facets. This procedure is left for future research.

E. SUMMARY

In this chapter, we developed new techniques for finding and lifting cuts from individual knapsack constraints. We described an algorithm that finds a maximally violated minimal cover cut for a knapsack constraint if such a cut exists, an algorithm that lifts a minimal cover cut to a lifted minimal cover cut and we developed a new lifting procedure called "interior lifting" that lifts a lifted minimal cover cut to a strong cover cut. We adapted polynomial-time facet finding algorithms to tighten strong cover cuts to violated facets, and identified conditions and created procedures that streamline the facet-finding process.

We identified a new type of knapsack cut, the "non-minimal cover cut," a cut that cannot be found by solving the traditional minimal cover separation problem. We developed non-minimal cover cut-finding procedures and a new lifting procedure called "deficit lifting" that creates a violated extended minimal cover valid inequality from a non-minimal cover cut, if necessary. The complete procedure of cut-finding, lifting and tightening for either a minimal or a non-minimal cover cut produces a violated facet in pseudo-polynomial time.

IV. KNAPSACK CUT EXTENSIONS

This chapter develops two extensions to the results of Chapter III on knapsack cuts. In particular, we show how to generate cuts for "elastic knapsack constraints" (knapsack constraints with penalized violation allowed), and for knapsack constraints with senses of "greater-than-or-equal-to" and "equality."

A. CUTS FOR ELASTIC KNAPSACK CONSTRAINTS

Our research extends the basic results on the knapsack polytope to include the generation of cuts from elastic knapsack constraints. The elastic knapsack constraint is

$$\sum_{j \in N} a_j x_j - z \leq b \tag{IV.1}$$

where the a_j and b are positive integers, $a_j \leq b \; \forall \; j \in N$, x_j are binary, and $0 \leq z \leq \overline{z}$; if z is not explicitly bounded above, then implicitly $\overline{z} = \sum_{j \in N} a_j - b$. The variable z represents the additional units of resource that can be used above and beyond the basic limit of b if an appropriate (linear) penalty is paid. We develop a method to cut off fractional solutions to elastic knapsack constraints that parallels many of the knapsack cut procedures developed in Chapter III. Because all cuts must be valid inequalities, we first examine valid inequalities for the elastic knapsack constraint.

1. Valid Inequalities for the Elastic Knapsack Polytope

We again focus on minimal covers, because we can extend many of our minimal cover cut results for standard knapsack constraints to elastic knapsack constraints. When we generate "elastic minimal cover valid inequalities" (EMCVI) for (IV.1), we note that if z is not explicitly bounded above, the implicit upper bound of $\overline{z} = \sum_{j \in N} a_j - b$ allows *any* integer solution to be selected for the right price. Thus, when $\overline{z}$ is implicit, z must appear in every EMCVI.

The conditions for the minimal cover are altered from the standard MCVI by the addition of the variable z. S given $\widehat{z}$, denoted $S_{\widehat{z}}$, will be an "elastic minimal cover" for (IV.1) if

$$\sum_{j \in S_{\widehat{z}}} a_j > b + \lfloor \widehat{z} \rfloor$$

and

$$\sum_{j \in S_{\widehat{z}} - \{i\}} a_j \leq b + \lfloor \widehat{z} \rfloor \,, \; \forall \, i \in S_{\widehat{z}}.$$

Thus, an EMCVI for (IV.1) is

$$\sum_{j \in S} x_j - \frac{z}{\sum_{j \in S} a_j - b} \leq |S| - 1.$$

Example 1 (part a) Consider the elastic knapsack constraint

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 + x_6 - z \leq 7 \tag{IV.2}$$

where $\overline{z} = 3$. Given an LP solution with $\widehat{z} = 2$, an elastic minimal cover is $S_{\widehat{z}} = \{1, 2, 3, 6\}$. The associated EMCVI is

$$x_1 + x_2 + x_3 + x_6 - \frac{z}{3} \leq 3. \tag{IV.3}$$

By rearranging (IV.3), it becomes clear that, when $\widehat{z} = \overline{z} = 3$,

$$x_1 + x_2 + x_3 + x_6 \leq 3 + \frac{z}{3}$$

is really enforcing

$$x_1 + x_2 + x_3 + x_6 \leq 4,$$

allowing all the variables in the valid inequality to be 1. When $\widehat{z} = 3$, the rearranged elastic constraint (IV.2)

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 + x_6 \leq 7 + z$$

is actually enforcing

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 + x_6 \leq 10$$

which does allow $x_1 = x_2 = x_3 = x_6 = 1$. This verifies that the EMCVI we developed is a "valid" inequality, as it does not disallow any valid solutions given $0 \leq \widehat{z} \leq \overline{z}$.

End example 1 (part a)

2. Finding a Maximally Violated Elastic Minimal Cover Cut

We now develop a procedure to find an elastic minimal cover cut (EMCC). A "maximally violated elastic minimal cover cut" (XVEMCC) for an elastic knapsack cut may be found by modifying the XVMCC-finding procedure found in Chapter III. Specifically, the objective function and constraints must be modified to account for the constraint violation variable z.

a. Cut-finding problem formulation

In order to find an XVEMCC, we must identify a knapsack constraint of the MIP that has at least one variable with a fractional LP solution value $\widehat{x}_j$. Once a suitable constraint is found, we solve the cut-finding problem:

Indices:

j $\in N = \{1, 2, 3, ..., n\}$ variable index for the knapsack constraint
N^+ $= \{ j \in N \mid \hat{x}_j > 0\}$ where $\hat{\mathrm{x}}$ is the solution of the LP relaxation of the MIP

Given data:

$\hat{x}_j$ value for x_j in a solution of the LP relaxation of the MIP
$\widehat{z}$ value for z in a solution of the LP relaxation of the MIP
a_j knapsack constraint coefficient of x_j
b right-hand side of the knapsack constraint

Decision variables:

h_j 1 if x_j is placed in the minimal cover inequality; 0 otherwise
r is a general integer variable that is the right-hand side of the XVMCC

Formulation:

$$
\begin{array}{lllllll}
z^*_{el} = & \text{maximize} & \sum\limits_{j\in N^+} \hat{x}_j h_j & -\frac{\hat{z}}{t-b} & -r & & \\
& \text{subject to:} & \sum\limits_{j\in N^+} a_j h_j & & & \geq & b + \lfloor \hat{z} \rfloor + 1 \\
& & \sum\limits_{j\in N^+} a_j h_j & & & \leq & b + \lfloor \hat{z} \rfloor + a_{\min} \\
& & \sum\limits_{j\in N^+} a_j h_j & -t & & = & 0 \\
& & \sum\limits_{j\in N^+} h_j & & -r & = & 1 \\
& & h_j & & & \in & \{0,1\} \ \forall\, j \in N^+ \\
& & & t & & \in & \{b + \lfloor \hat{z} \rfloor + 1, ..., b + \lfloor \hat{z} \rfloor + a_{\min}\} \\
& & & & r & \in & \{0, 1, 2, ..., |N^+| - 1\}
\end{array}
\tag{IV.4}
$$

where $a_{\min} = \min\limits_{j \in N^+} a_j$.

To ensure that all EMCCs are found, we actually solve the cut-finding problem $|N^+|$ times with each $a_j = a_{\min}$, $j \in N^+$ and requiring $h_j \equiv 0$ for every $a_j < a_{\min}$. (Actually, we need to solve the cut-finding problem only $|N^+| - p$ times, where$|N^+| - p$ is the number of unique a_j, $j \in N^+$.)

Although the above IP is nonlinear, it can be solved fairly easily, and an EMCC has been found if

$$\sum_{j\in S_{\hat{z}}} \hat{x}_j h^*_j - \frac{\hat{z}}{t^* - b} - r^* > 0,$$

where $\mathbf{h}^*, t^*, r^*$ solve (IV.4).

b. Dynamic-programming reformulation

We reformulate the cut-finding problem in order to solve it with dynamic programming as we did in Chapter III. Thus, the actual cut-finding problem we solve is: xxxxx

$$
\begin{array}{lllllll}
z^*_{el} = 1 + & \max\limits_t \max\limits_h & \sum\limits_{j\in N^+} (\hat{x}_j - 1) h_j & -\frac{\hat{z}}{t-b} & & & \\
& \text{subject to:} & b' \leq \sum\limits_{j\in N^+} a_j h_j & & & \leq & b'' \\
& & \sum\limits_{j\in N^+} a_j h_j & -t & & = & 0 \\
& & h_j & & & \in & \{0,1\} \ \forall\, j \in N^+ \\
& & & t & & \in & \{b', ..., b''\},
\end{array}
\tag{IV.5}
$$

where $b' = b + \lfloor \widehat{z} \rfloor + 1$ and $b'' = b + \lfloor \widehat{z} \rfloor + a_{\min}$. The problem is solved at most $|N^+|$ times with each unique $a_j = a_{\min}$, $j \in N^+$ and requiring every $a_j < a_{\min}$, $j \in N^+$ to have $h_j \equiv 0$.

The reformulation (IV.5) can be solved via a DP recursion
DP Recursion 3

Initial conditions:
$d_0(0) = 1$
$d_0(a') = -\infty$ for $a' = 1, ..., b + a_{\min}$
$d_k(a') = -\infty$ for all k, $a' < 0$

Recursion:
$$d_k(a') = \max_h \{d_{k-1}(a'), d_{k-1}(a' - a_k) + (\widehat{x}_k - 1)\}$$
$$\text{for } k = 1, ..., |N^+|, \; a' = 0, ..., b + a_{\min}.$$

The solution to the problem is

$$z^*_{el} = 1 + \max_{b' \le a' \le b''} d_{|N^+|}(a') - \frac{\widehat{z}}{a' - b}.$$

If $z^*_{el} > 0$, where h* solves (IV.5), an XVEMCC has been found, and the corresponding elastic minimal cover $S_{\widehat{z}} = \{k | h^*_k = 1\}$ can be recovered through auxiliary data structures within the dynamic programming algorithm. The XVEMCC is

$$\sum_{j \in S_{\widehat{z}}} x_j - \frac{z}{\sum_{j \in S_{\widehat{z}}} a_j - b} \le |S_{\widehat{z}}| - 1.$$

Example 1 (part b) Suppose the solution to the LP relaxation of a MIP with knapsack constraint

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 + x_6 - z \le 7$$

is

$$\hat{x}_1 = 1, \hat{x}_2 = 1, \hat{x}_3 = \frac{3}{4}, \hat{x}_4 = 0, \hat{x}_5 = 0, \hat{x}_6 = 1, \widehat{z} = 2.$$

Recall the modification to the cut-finding problem changes the constraint, replacing the original b with $b + \lfloor \widehat{z} \rfloor$:

$$2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 + x_6 \le 9.$$

The cut-finding problem looks for the most violated minimal cover cut, with

$$\text{violation} = \sum_{j \in S_{\hat{z}}} \hat{x}_j - \frac{\hat{z}}{\sum_{j \in S_{\hat{z}}} a_j - b} - |S_{\hat{z}}| - 1.$$

We solve (IV.5) and find the EMCC

$$x_1 + x_2 + x_3 + x_6 - \frac{z}{3} \leq 3$$

which is violated, since

$$\begin{aligned} \text{violation} &= \sum_{j \in S_{\hat{z}}} \hat{x}_j - \frac{\hat{z}}{\sum_{j \in S_{\hat{z}}} a_j - b} - (|S_{\hat{z}}| - 1) \\ &= 3\frac{3}{4} - \frac{2}{3} - 3 \\ &= \frac{1}{12}. \end{aligned}$$

End example 1 (part b)

3. Lifting and Tightening Elastic Minimal Cover Cuts

Simple lifting can be conducted just as with standard knapsack cuts, but interior lifting and tightening must be altered in order to ensure that the cuts remain violated and valid.

a. Simple lifting

Any variable with a constraint coefficient greater than or equal to the largest constraint coefficient of the variables in the minimal cover S can be lifted.

Example 1 (part c) Recall the EMCC from part b:

$$x_1 + x_2 + x_3 + x_6 - \frac{z}{3} \leq 3.$$

Because $a_4 > a_3 = \max_{j \in S} a_j$, the variable x_4 is lifted, creating the lifted EMCC

$$x_1 + x_2 + x_3 + x_4 + x_6 - \frac{z}{3} \leq 3.$$

End example 1 (part c)

b. Interior lifting

As with standard knapsack valid inequalities, it is theoretically possible to lift an EMCC to an elastic strong cover cut. However, once interior lifting is completed, the coefficient of the variable z in the EMCC must be recomputed using the coefficients of the variables in the strong cover $S'_{\hat{z}}$. Since the cover is now strong, $\sum_{j \in S'_{\hat{z}}} a_j < \sum_{j \in S_{\hat{z}}} a_j$, and the coefficient of z, $\frac{1}{\sum_{j \in S'_{\hat{z}}} a_j - b}$, is now larger. Because the z term is subtracted when calculating the amount of violation,

$$\text{violation} = \sum_{j \in S_{\hat{z}}} \hat{x}_j - \frac{\hat{z}}{\sum_{j \in S_{\hat{z}}} a_j - b},$$

the adjustment of the coefficient can cause the EMCC to "lose" its violation. Therefore, in practice, we do not use interior lifting on EMCCs that have $\hat{z} > 0$.

Example 2 (part a) Suppose the solution to the LP relaxation of a MIP with elastic knapsack constraint

$$20x_1 + 25x_2 + 25x_3 + 45x_4 + 21x_5 - z \leq 50$$

is

$$\hat{x}_1 = 1, \hat{x}_2 = 1, \hat{x}_3 = \frac{5}{9}, \hat{x}_4 = 0, \hat{x}_5 = 0, \hat{z} = 8\frac{8}{9}.$$

The knapsack constraint contains two fractional variables. The solution to (IV.5) is the EMCC with $S_{\hat{z}} = \{1, 2, 3\}$:

$$x_1 + x_2 + x_3 - \frac{z}{20} \leq 2$$

which is a cut because

$$\hat{x}_1 + \hat{x}_2 + \hat{x}_3 - \frac{\hat{z}}{20} > 2.$$

Simple lifting extends the EMCC to

$$x_1 + x_2 + x_3 + x_4 - \frac{z}{20} \leq 2,$$

which is still violated since $\hat{x}_4 = 0$. Next, interior lifting allows us to include the last

variable and create an elastic strong cover valid inequality. However, we must recompute the coefficient of z. When we do that, the valid inequality

$$x_1 + x_2 + x_3 + x_4 + x_5 - \frac{z}{16} \leq 2$$

is not a cut because

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 + \widehat{x}_4 + \widehat{x}_5 - \frac{\widehat{z}}{16} = 2.$$

End example 2 (part a)

c. Facets of the elastic knapsack polytope

Once we have found an EMCC, we would like to form a facet of the elastic knapsack polytope. The variable z complicates matters, as we have already experienced with interior lifting. Since we cannot blindly apply the results from Chapter III, we must carefully identify under which conditions we can create a facet of the elastic knapsack polytope.

When the elastic minimal cover $S_{\widehat{z}}$ is strong, and $\overline{z} = \sum_{j \in N} a_j - b$, we can lift the elastic strong cover cut to a facet of the elastic knapsack polytope. The conditions that identify if a cover $S_{\widehat{z}}$ is strong are

$$E(S_{\widehat{z}}) = N, \tag{IV.6}$$

or, if there are no variables that could be lifted (with a coefficient of 1) through interior lifting, that is, if

$$\sum_{j \in S_{\widehat{z}} - \{j_{\max}\}} a_j + a_i \leq b + \lfloor \widehat{z} \rfloor \,, \ \forall i \in N - E(S_{\widehat{z}}), \tag{IV.7}$$

where $j_{\max} = \arg\max_{j \in S_{\widehat{z}}} a_j$. If either condition (IV.6) or condition (IV.7) is met, then we redesignate $S_{\widehat{z}}$ as the elastic strong cover $S'_{\widehat{z}}$, and

$$\sum_{j \in S'_{\widehat{z}}} x_j + \sum_{j \in E(S'_{\widehat{z}}) - S'_{\widehat{z}}} x_j - \frac{z}{\sum_{j \in S'_{\widehat{z}}} a_j - b} \leq |S'_{\widehat{z}}| - 1$$

is a facet of the elastic knapsack polytope.

Theorem 4.1 *If the elastic minimal cover $S'_{\widehat{z}}$ is strong, then the elastic strong cover valid*

inequality

$$\sum_{j\in S'_{\widehat{z}}} x_j + \sum_{j\in E(S'_{\widehat{z}})-S'_{\widehat{z}}} x_j - \frac{z}{\sum_{j\in S'_{\widehat{z}}} a_j - b} \leq |S'_{\widehat{z}}| - 1, \tag{IV.8}$$

where $\overline{z} = \sum_{j\in N} a_j - b$ *and* $a_j \leq b \; \forall \; j \in N$, *is a facet of the elastic knapsack polytope*

$$P_E = conv\left\{ \mathbf{x} \in \{0,1\}^n, 0 \leq z \leq \overline{z} \,|\sum_{j\in N} a_j x_j - z \leq b \right\}. \tag{IV.9}$$

Proof. The valid inequality (IV.8) is a facet of the elastic knapsack polytope if it is satisfied by every $\mathbf{x} \in P_E$, and satisfied at equality by exactly d affinely independent points $\mathbf{x} \in P_E$, where $d = n + 1$ is the dimension of P_E. Consider the elastic constraint

$$\sum_{j\in N} a_j x_j - z \leq b, \tag{IV.10}$$

where $a_j \leq b \; \forall \; j \in N$, $a_1 + a_2 > b + \widehat{z}$, $n = |N|$ and z is not explicitly bounded.

Since $S'_{\widehat{z}}$ is strong, a strong elastic minimal cover valid inequality for (IV.10) is

$$\sum_{j\in S'_{\widehat{z}}} x_j + \sum_{j\in E(S'_{\widehat{z}})-S'_{\widehat{z}}} x_j - \frac{z}{\sum_{j\in S'_{\widehat{z}}} a_j - b} \leq |S'_{\widehat{z}}| - 1, \tag{IV.11}$$

where $\overline{z} = \sum_{j\in N} a_j - b$.

x_1	1	0	1	1	1	$\cdots$	1	1
x_2	0	1	1	1	1	$\cdots$	1	1
z	0	0	Σ_2	Σ_3	Σ_4	$\cdots$	Σ_{n-1}	Σ_n
x_3	0	0	0	1	0	$\cdots$	0	0
x_4	0	0	0	0	1	$\ddots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\ddots$	$\vdots$
x_{n-1}	0	0	0	0	0	$\ddots$	1	0
x_n	0	0	0	0	0	$\cdots$	0	1

Table 1. This table contains the n linearly independent points that demonstrate the elastic strong cover valid inequality (where $|S'_{\widehat{z}}| = 2$) is a facet of the elastic knapsack polytope for the strong cover $S'_{\widehat{z}}$ when $E(S'_{\widehat{z}}) = N$. $\Sigma_2 = a_1 + a_2 - b$ and $\Sigma_k = a_1 + a_2 + a_k - b$, $k = 3, 4, ..., n$.

In Table 1 there are n linearly independent points that satisfy (IV.11) at equality, and since linear independence implies affine independence, the dimension of (IV.11) is at least n. Since the dimension of the elastic polytope P_E is $n + 1$, (IV.11) is a facet of P_E.

The result for $|S'_{\hat{z}}| = k$, $k > 2$ follows by induction on k.

Note: We can easily form $|N - E(S'_{\hat{z}})|$ other linearly independent points for the case $E(S'_{\hat{z}}) < N$ to verify that the facet has appropriate dimension when variables are implicitly assigned a facet coefficient of 0 by their exclusion from the inequality. *QED*

When we have an EMCC and $\widehat{z} > 0$, we must forego interior lifting. Thus, if $S_{\hat{z}}$ is not strong, we will not create a strong cover from the elastic minimal cover, and the (possibly lifted) EMCC may not be a facet. This next example shows why this is true.

Example 3 Suppose the solution to the LP relaxation of a MIP with elastic knapsack constraint (with z not explicitly bounded)

$$20x_1 + 25x_2 + 21x_3 - z \leq 40$$

is

$$\hat{x}_1 = 1, \hat{x}_2 = \frac{21}{25}, \hat{x}_3 = 0, \widehat{z} = 1.$$

The knapsack constraint contains one fractional variable. The solution to (IV.5) is the EMCC with $S_{\hat{z}} = \{1, 2\}$:

$$x_1 + x_2 - \frac{z}{5} \leq 1 \tag{IV.12}$$

which is a cut because

$$\widehat{x}_1 + \widehat{x}_2 - \frac{\widehat{z}}{5} > 1.$$

When we include x_3 by interior lifting,

$$x_1 + x_2 + x_3 - \frac{z}{1} \leq 1$$

is not a cut because

$$\widehat{x}_1 + \widehat{x}_2 + \widehat{x}_3 - \frac{\widehat{z}}{1} < 1.$$

So, we must use (IV.12) as a cut. But, is it a facet of the elastic polytope? Recall that a facet of the integer polytope P_E (IV.9) is an inequality that is satisfied by every $\mathrm{x} \in P_E$, and satisfied at equality by exactly d affinely independent points $\mathrm{x} \in P_E$, where d is the

dimension of P_E. We examine the points that satisfy (IV.12) at equality in this table:

	$x \in P_E$			$x \notin P_E$		
x_1	1	0	1	1	0	1
x_2	0	1	1	0	1	1
z	0	0	5	0	0	5
x_3	0	0	0	1	1	1

Note the points that satisfy (IV.12) at equality, but are *not* in the polytope P_E. Since we only have 3 points $x \in P_E$ that satisfy (IV.12) at equality, and the dimension of P_E is 4, we cannot possibly have four affinely independent points that satisfy (IV.12), so (IV.12) cannot be a facet.

End example 3

d. Tightening

In the previous section, we developed a procedure to generate strong cuts and facets for elastic knapsack constraints with an implicit upper bound $\overline{z} = \sum_{j \in N} a_j - b$ on the constraint violation variable z. When $\overline{z} < \sum_{j \in N} a_j - b$, the elastic knapsack constraint will not allow all variables to be set to 1 at the same time, and some of the variables lifted into the EMCC may require coefficients greater than one. We can use the tightening Procedures 3 or 4 from Chapter III by deriving the appropriate coefficients ω_j for variables x_j as if they belonged to the knapsack constraint

$$\sum_{j \in N} a_j x_j \leq b + \overline{z}.$$

To use Procedure 3, the elastic cover must be tight, that is

$$\sum_{j \in S_{\overline{z}}} a_j = b + \overline{z} + 1.$$

If the elastic cover is not tight, Procedure 4 must be used. To use either procedure, substitute $b + \overline{z}$ for b. The EMCC is then tightened to

$$\sum_{j \in S'_{\overline{z}}} x_j + \sum_{j \in E(S'_{\overline{z}}) - S'_{\overline{z}}} \omega_j x_j - \frac{z}{\sum_{j \in S'_{\overline{z}}} a_j - b} \leq |S'_{\overline{z}}| - 1,$$

which is a strong cut (perhaps a facet) for the elastic knapsack polytope.

We can also use these tightening procedures when $S_{\hat{z}}$ is not strong by ignoring any variables $x_i, i \in N - E(S_{\hat{z}})$ that do not meet condition (IV.7). But, in this case, we know that we will not form a facet.

Example 2 (part b) Given the knapsack constraint

$$20x_1 + 25x_2 + 25x_3 + 45x_4 + 21x_5 - z \leq 50,$$

and $\overline{z} = 10$, we tighten the cut formed in Example 2, part a,

$$x_1 + x_2 + x_3 + x_4 - \frac{z}{20} \leq 2,$$

by rewriting the constraint with the substitution $b + \overline{z}$ for b :

$$20x_1 + 25x_2 + 25x_3 + 45x_4 + 21x_5 \leq 50 + \overline{z} = 60.$$

Since $\sum_{j \in S_{\hat{z}}} a_j > b + \overline{z} + 1$, we use the Procedure 4 lifting process in Chapter III. Thus, we assign $\omega_4 = 2$, and the new valid inequality is

$$x_1 + x_2 + x_3 + 2x_4 - \frac{z}{20} \leq 2.$$

End example 2 (part b)

Note that we ignored the variable x_5, the variable that, through interior lifting, removed the violation. Our lifting process is still valid, but we forego the possibility of finding a facet of the elastic knapsack polytope.

4. Non-Standard Elastic Knapsack Constraints

We consider a variant of (IV.1) where one or more $a_j > b, j \in N$. When an elastic minimal cover is found with one of the variables having such a constraint coefficient, the valid inequality (and cut) derived is altered. Define

$$S_{\hat{z}}^{>} = \{j | a_j > b, j \in S_{\hat{z}}\},$$

and let $a^{>}_{j_{\min}} = \min_{j \in S^{>}_{\hat{z}}} a_j$. For any elastic minimal cover containing such a variable, the valid inequality is

$$\sum_{j \in S'_{\hat{z}}} x_j + \sum_{j \in E(S'_{\hat{z}}) - S'_{\hat{z}}} \omega_j x_j - \frac{z}{a^{>}_{j_{\min}} - b} \leq |S'_{\hat{z}}| - 1,$$

where the coefficient of z has changed from $\frac{1}{\sum_{j \in S'_{\hat{z}}} a_j - b}$ to $\frac{1}{a^{>}_{j_{\min}} - b}$. The cut-finding problem must be amended to reflect this coefficient change as well.

We also can derive cuts for another elastic constraint,

$$\sum_{j \in J} a_j x_j + z \geq b.$$

After a simple conversion, discussed in the next section, the constraint becomes of form (IV.1), and all procedures in this section can be applied.

B. KNAPSACK CONSTRAINTS WITH NON-STANDARD SENSES

All our results thus far have been for the knapsack polytope associated with the "standard" form

$$\sum_{j \in N} a_j x_j \leq b. \tag{IV.13}$$

Our results are easily adapted for knapsack constraints

$$\sum_{j \in N} a_j x_j \geq b \tag{IV.14}$$

and

$$\sum_{j \in N} a_j x_j = b$$

where a_j and b are positive integers, x_j are binary variables, and $N = \{1, ..., n\}$.

1. Greater-Than-Or-Equal-To Knapsack Constraints

We use our existing algorithms by simply converting (IV.14) to the form of (IV.13). This is accomplished by replacing the variable x_j with its complement $(1 - y_j)$. (Crowder, et al. (1983) substitute complementing variables to convert less-than-or-equal-to knapsack

constraints with one or more negative coefficients a_j to (IV.13).) Thus, by the substitution in (IV.14), we have

$$\sum_{j \in N} a_j(1 - y_j) \geq b,$$

the new standard knapsack constraint is

$$\sum_{j \in N} a_j y_j \leq \sum_{j \in N} a_j - b \tag{IV.15}$$

where a_j and $\sum_{j \in N} a_j - b$ are positive integers,the y_j are binary variables, and $N = \{1, ..., n\}$.

Once (IV.15) is formed, we find the complementary LP solution values $\widehat{y}_j = 1 - \widehat{x}_j$. We then apply one of the cut-finding problems and, if an XVMCC or an NMCC exists, all lifting and tightening procedures are applied and we find a facet

$$\sum_{j \in S'} y_j + \sum_{j \in \widetilde{S}'} \omega_j y_j \leq |S'| - 1.$$

We now transform the facet back into the original variable space before appending it to the MIP. The facet added to the MIP is

$$\sum_{j \in S'} x_j + \sum_{j \in \widetilde{S}'} \omega_j x_j \geq \sum_{j \in \widetilde{S}'} \omega_j + 1.$$

Example 4 Suppose the solution to the LP relaxation of a MIP with knapsack constraint

$$4x_1 + 4x_2 + 2x_3 + 2x_4 + 5x_5 + 3x_6 + 9x_7 + 3x_8 + 8x_9 \geq 30$$

is

$$\widehat{x}_1 = \frac{3}{4}, \widehat{x}_2 = 0, \widehat{x}_3 = 0, \widehat{x}_4 = 0, \widehat{x}_5 = \frac{4}{5}, \widehat{x}_6 = 1, \widehat{x}_7 = 1, \widehat{x}_8 = 1, \widehat{x}_9 = 1.$$

The knapsack constraint has two fractional variables. The transformed knapsack constraint

$$4y_1 + 4y_2 + 2y_3 + 2y_4 + 5y_5 + 3y_6 + 9y_7 + 3y_8 + 8y_9 \leq 10$$

has a transformed LP solution of

$$\widehat{y}_1 = \frac{1}{4}, \widehat{y}_2 = 1, \widehat{y}_3 = 1, \widehat{y}_4 = 1, \widehat{y}_5 = \frac{1}{5}, \widehat{y}_j = 0 \; \forall \; j > 5$$

The solution to the XVMCC-finding problem is an XVMCC with minimal cover $S = \{1, 2, 3, 4\}$

$$y_1 + y_2 + y_3 + y_4 \leq 3.$$

Applying Procedures 1, 2 and 3 yield the facet

$$y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + 3y_7 + 2y_9 \leq 3.$$

Transforming back to the original variable space yields the facet

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 3x_7 + 2x_9 \geq 8$$

which is now appended to the MIP to cut off $\widehat{\mathrm{x}}$.

End example 4

2. Equality Knapsack Constraints

We can use our existing algorithms for knapsack constraints with equality senses, also. First, we convert

$$\sum_{j \in N} a_j x_j = b$$

to

$$\sum_{j \in N} a_j x_j \leq b \tag{IV.16}$$

and

$$\sum_{j \in N} a_j x_j \geq b. \tag{IV.17}$$

Then we convert (IV.17) by complementing, and apply all applicable procedures from Chapter III to both constraints that are now of the form (IV.16).

Example 5 Suppose the solution to the LP relaxation of a MIP with knapsack constraint

$$2x_1 + 2x_2 + 3x_3 + 4x_4 = 5$$

is

$$\widehat{x}_1 = 1, \widehat{x}_2 = 1, \widehat{x}_3 = \frac{1}{3}, \widehat{x}_4 = 0.$$

We can represent the constraint by

$$2x_1 + 2x_2 + 3x_3 + 4x_4 \leq 5 \tag{IV.18}$$

and

$$2x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5. \tag{IV.19}$$

The lifted and tightened minimal cover cut for (IV.18) is

$$x_1 + x_2 + x_3 + 2x_4 \leq 2.$$

We transform (IV.19) by complementing, which yields

$$2y_1 + 2y_2 + 3y_3 + 4y_4 \leq 6, \tag{IV.20}$$

with a transformed LP solution of

$$\widehat{y}_1 = 0, \widehat{y}_2 = 0, \widehat{y}_3 = \frac{2}{3}, \widehat{y}_4 = 1.$$

A minimal cover cut for (IV.20) is

$$y_3 + y_4 \leq 1.$$

Transforming back to the original variable space yields the cut

$$x_3 + x_4 \geq 1.$$

Both cuts are now appended to the MIP to cut off $\widehat{\mathbf{x}}$.

End example 5

V. EXPLICIT-CONSTRAINT BRANCHING

In this chapter we develop the new technique of "explicit-constraint branching" (ECB) to improve the branch-and-bound portion of composite enumeration. Special Ordered Sets and Foster and Ryan's branching technique for overlapping set-partitioning constraints (Foster and Ryan, 1981) demonstrate how constraint branching can improve the solution process for MIPs, but these implicit-constraint-branching techniques require a special problem structure. ECB adds structure to a MIP by adding new constraints and new integer variables and does not require any special relationships within the problem structure. Like implicit-constraint branching, the goal of ECB is to moderate the branching process by more "evenly" partitioning the feasible region of the LP relaxation of the MIP. The value of this simple technique is demonstrated by empirical evidence presented in Chapter VI.

A. INTRODUCTION

ECB adds one or more constraints of the form

$$\sum_{j \in J_i} a_j x_j - y_i = 0 \tag{V.1}$$

to a MIP where J is the set of indices of integer variables of the MIP, a_j are integer constants, $J_i \subseteq J$, and the y_i are general integer "branching" variables. For reasons discussed later, we assume $a_j \equiv 1$ for all $j \in J_i$ in the following discussion.

Each ECB constraint is created (with $a_j = 1 \; \forall \; j \in J_i$) to allow the branch-and-bound process to branch on a sum of integer variables before branching on any individual integer variable in the sum; intuitively the sum should be integer before all of the individual variables in the sum need be integer. We implement this by setting the "branching priority" higher for the branching variable y_i than the x_j in the branch-and-bound algorithm. (Modern integer-programming solvers usually allow this branch-and-bound option.) Then, whenever branch and bound sees an intermediate solution with some non-integer $\widehat{y}_i$, branching derives

the two restrictions for the MIP at the current node

$$y_i \leq \lfloor \widehat{y}_i \rfloor \text{ or } y_i \geq \lfloor \widehat{y}_i \rfloor + 1.$$

Branch and bound will only derive restrictions from a fractional $\widehat{x}_j$ when all $\widehat{y}_i$ are integer, continuing to do so unless some $\widehat{y}_i$ becomes non-integer again.

The following series of examples demonstrate how ECB might be applied. First, we repeat the example of Chapter I to illustrate ECB where only a single ECB constraint is added, and to illustrate the potential effectiveness of the technique.

Example 1. Recall the binary IP of Example 2, Section I.3:

$$\begin{array}{lll} \text{maximize} & \sum\limits_{j \in J} x_j & \\ \text{subject to:} & \sum\limits_{j \in J} 2x_j & \leq \; 2\left\lfloor \frac{|J|}{2} \right\rfloor + 1 \\ & x_j & \in \; \{0,1\} \; \forall j \in J. \end{array}$$

Variable-based branch and bound forms a partition based on the one fractional variable, which we designate as $\widehat{x}_{j_f}$, and derives the restrictions

$$x_{j_f} \leq 0 \text{ or } x_{j_f} \geq 1.$$

The process of fixing variables and fathoming nodes continues until all $\binom{|J|}{\lfloor \frac{|J|}{2} \rfloor}$ alternate optimal solutions are found and the first is declared optimal.

We reduce the amount of nodes branch and bound must enumerate by adding an ECB constraint and creating the problem

$$\begin{array}{lllll} \text{maximize} & \sum\limits_{j \in J} x_j & & & \\ \text{subject to:} & \sum\limits_{j \in J} 2x_j & & \leq & 2\left\lfloor \frac{|J|}{2} \right\rfloor + 1 \\ & \sum\limits_{j \in J} x_j & -y & = & 0 \\ & x_j & & \in & \{0,1\} \; \forall \, j \in J \\ & & y & \in & \{0,1,2,...,|J|\}. \end{array}$$

The variable-based branch-and-bound partition based on $\widehat{y}$ derives the restrictions

$$y \leq \left\lfloor \frac{|J|}{2} \right\rfloor \text{ or } y \geq \left\lfloor \frac{|J|}{2} \right\rfloor + 1,$$

and branch and bound evaluates only three LPs before the problem is solved. Thus, by branching on the sum of integer variables, we solved the IP without fixing any binary variables.

End Example 1

Another example demonstrates how a single ECB constraint might be used with set-packing, set-covering, and set-partitioning problems, and hybrid versions of these problems.

Example 2.

part (a). Consider an IP,

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in J} c_j x_j \\ \text{subject to:} \quad & \sum_{j \in J} a_{ij} x_j \quad \begin{matrix} \leq \\ \geq \end{matrix} \quad 1 \; \forall \, i = 1, ..., m \\ & x_j \quad \in \quad \{0, 1\}, \; j \in J, \end{aligned}$$

where the $a_{ij} \in \{0, 1\}$, constraints with a sense of $\leq$ are set-packing constraints, constraints with a sense of $\geq$ are set-covering constraints, and constraints with a sense of $=$ are set-partitioning constraints. An obvious requirement for any solution of this class of problems is that the sum of all binary variables be integer. This requirement can be satisfied in a branch-and-bound algorithm (before requiring all binary variables to be integer) by adding the construct

$$\sum_{j \in J} x_j - y = 0,$$

where $y \geq 0$, and integer. Again, the branching priority is set higher for y than for the x_j.

part (b).

A specific example demonstrates how one ECB constraint can reduce the work of

the branch-and-bound algorithm in a set-packing IP. Consider the set-packing IP:

$$\begin{array}{lrcl}
\text{maximize} & \sum_{k=1}^{K} (x_{k_1} + x_{k_2} + x_{k_3}) & & \\
\text{subject to:} & \sum_{k=1}^{K} (x_{k_1} \qquad + x_{k_3}) & \leq & 1 \\
& \sum_{k=1}^{K} (x_{k_1} + x_{k_2} \qquad) & \leq & 1 \\
& \sum_{k=1}^{K} (\qquad x_{k_2} + x_{k_3}) & \leq & 1 \\
& x_{k_1}, \quad x_{k_2}, \quad x_{k_3}, & \in & \{0,1\}, \; k = 1, ..., K.
\end{array}$$

This problem consists of a set of K "odd cycles" of length three (e.g., Hoffman and Padberg, 1993). Without loss of generality, we may assume that an optimal LP extreme point solution will be of the form

$$x_{1_1} = x_{1_2} = x_{1_3} = \frac{1}{2}, \; x_{k_1} = x_{k_2} = x_{k_3} = 0 \; \forall \; k > 1.$$

It is relatively straightforward to see that the best-case branch-and-bound scenario must solve approximately K LPs, and the worst-case scenario must solve approximately $3K$ LPs before a declared optimal solution is found.

By adding an ECB constraint

$$\sum_{k=1}^{K} (x_{k_1} + x_{k_2} + x_{k_3}) - y = 0$$

and branching on y first, our initial separation will be

$$\sum_{k=1}^{K} (x_{k_1} + x_{k_2} + x_{k_3}) \leq 1 \text{ or } \sum_{k=1}^{K} (x_{k_1} + x_{k_2} + x_{k_3}) \geq 2.$$

The LP resulting from the second constraint is infeasible, and the LP resulting from the first constraint finds a declared optimal solution to the problem, since all feasible LP extreme points have some $x_{k_i} = 1$, and all the other variables 0.

Larger problems with sets of these constraints embedded in the constraint set can be handled with cuts of the form

$$x_{k_1} + x_{k_2} + x_{k_3} \leq 1 \; \forall \; k = 1, ..., K$$

but this entails first searching for and finding all sets of odd cycles. ECB deals with this issue without such overhead.

End Example 2

Example 3 Consider an IP for a capacitated plant location problem:

$$
\begin{array}{lrcccl}
\text{minimize} & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} & + & \sum_{i=1}^{n} f_i z_i & & \\
\text{subject to:} & \sum_{i=1}^{n} x_{ij} & & & = \; 1 & \forall\, j \\
& \sum_{j=1}^{m} D_j x_{ij} & - & K_i z_i & \leq \; 0 & \forall\, i \\
& & & \sum_{i=1}^{n} z_i & = \; k & \\
& x_{ij} & - & z_i & \leq \; 0 & \forall\, i, \forall\, j \\
& x_{ij} & & & \in \; \{0,1\} & \forall\, i, \forall\, j \\
& & & z_i & \in \; \{0,1\} & \forall\, i
\end{array}
$$

where D_j is the demand associated with customer j and K_i is the capacity of the plant located at i. If $z_i = 1$, then a plant is located at i, and if $x_{ij} = 1$, then customer j's demand is satisfied by the plant located at i.

Before we form any ECB constraints, we set the branching priority higher for the plant location variables z_i than the branching priority for the customer variables x_{ij}. This ensures that branch and bound decides which plants are opened before deciding which individual customers will be served by a plant. (This is equivalent to the "propagation branching" described by Guignard and Spielberg (1977).) We also add the ECB constraints

$$\sum_{j=1}^{m} x_{ij} - y_i = 0 \; \forall\, i.$$

to ensure that the sum of the customers for each opened plant is integer before branching on any individual customer. Branching priorities p_w for variable w are set as follows: $p_{z_i} > p_{y_i} > p_{x_{ij}}$, where the largest priority is branched on first.

End Example 4

In similar problems where the number of facilities, such as plants, warehouses, or distribution centers is not fixed but is perhaps bounded (Geoffrion and Graves, 1974), it makes sense to add an ECB constraint to ensure that the number of facilities opened is in-

teger before branching on sums of customers. In our plant location example, if we replaced the requirement to open k plants with an upper bound of k instead, we could add the ECB constraint

$$\sum_{i=1}^{n} z_i - y = 0$$

where $y \in \{0, 1, 2, ..., k\}$, and assign branching priorities $p_y > p_{z_i} > p_{y_i} > p_{x_{ij}}$.

B. GENERAL EXPLICIT-CONSTRAINT BRANCHING

We define two different variants of ECB. "Basic ECB" allows a variable to appear at most once in an ECB constraint. The added constraints are

$$\sum_{j \in J_i} x_j - y_i = 0, \tag{V.2}$$

where the J_i are all disjoint subsets of J; the y_i are assigned higher branching priorities than the x_j.

"Nested ECB" uses the subsets of the variables from basic ECB constraints to add constraints of the form

$$\sum_{j \in J_{i_k}} x_j - y_{i_k} = 0, \ k = 1, 2, ..., m,$$

where $J_{i_k} \subset J_i$, and all J_{i_k} are disjoint. In this case, the y_i are assigned higher branching priorities than the y_{i_k}, which in turn are assigned higher priorities than the x_j. We can recursively form more nested constraints from each previously formed nested constraint ($J_{i_{kq}} \subset J_{i_k}, q = 1, 2, ..., Q$), so that we have multiple "levels" of nesting, and thus multiple levels of branching variables where the priorities are commensurately assigned.

Nested constraint branching could also be implemented using a "bottom up" approach. Let $J_1 \cup J_2 \cup ... \cup J_m = J$ define a partition of J, and create the bottom level of ECB constraints

$$\sum_{j \in J_k} x_j - y_k = 0, \ k = 1, 2, ..., m.$$

Then, form the next two levels

$$\sum_{k \in K_\ell} y_k - y'_\ell = 0,$$
$$\sum_{\ell \in L_p} y'_\ell - y''_p = 0,$$

and so on. Branching priorities would be determined so that $p_{x_j} < p_{y_k} < p_{y'_\ell} < p_{y''_p}$, etc. This bottom-up approach to nested ECB is not implemented in this dissertation.

C. CONSTRAINT-BRANCHING PROCEDURES

We distinguish between different applications of branching techniques. A *static* procedure adds branching constraints before the MIP solution procedure has begun. In practice, we decide the number and composition of ECB constraints to add for a static procedure by examining of the constraint set of the MIP. A *dynamic* procedure adds branching constraints during the branch-and-bound process, using information gleaned from solutions of LP relaxations at nodes of the enumeration tree. This procedure is similar to the cutting-plane procedure called "branch and cut." One such dynamic procedure has been used by Jörnsten and Värbrand (1991) to solve generalized assignment problems. These authors refer to the technique as "generalized branching" and credit Jörnsten and Larsson (1988). Implicit-constraint branching is another dynamic procedure that uses solution information from the LP relaxations in the branch and bound to partition the set of variables. We have developed "semi-dynamic ECB" that uses LP solution information to determine the composition of the added ECB constraints. We explain semi-dynamic ECB in the context of nested ECB in Section D. We consider only static and semi-dynamic procedures in this dissertation.

D. APPLICATIONS

1. Basic ECB

Problem structure will often dictate how basic static ECB constraints should be added. For instance, in the generalized assignment problem (GAP), defined in Chapter

II, there are knapsack constraints

$$\sum_{o \in O_t} h_{ot} x_{ot} \leq H_t,$$

$x_{ot} \in \{0, 1\}$, for each truck t. It is intuitively appealing to enforce the requirement that "the sum of order variables on a truck must be a general integer" before enforcing the requirement that "each order variable on a truck must be binary." Furthermore, there are many fewer of the former requirements to satisfy (typically, $|T| << \sum_t |O_t|$), and satisfying them first may achieve satisfaction of the latter requirements with little or no additional work. This enables us to solve, or nearly solve, the GAP by branching a modest number of times on a small number of general integer variables rather than branching many times on a large number of binary variables.

We implement basic ECB by adding the following constraints to the GAP:

$$\sum_{o \in O_t} x_{ot} - y_t = 0 \; \forall \; t \in T,$$

where the y_t are general integer variables. This modified GAP has $|T|$ new constraints that must be satisfied and $|T|$ new integer variables that must be solved for. We hope that the additional burden placed on the LP solver embedded in the branch-and-bound algorithm, (LP solvers operate, on average, in polynomial time), is outweighed by the potentially exponential reduction in the number of LPs that must be solved.

We could form the ECB constraints

$$\sum_{o \in O_t} h_{ot} x_{ot} - y_t = 0 \; \forall \; t \in T, \tag{V.3}$$

which correspond to our original definition (V.1), and treat y_t as a general integer branching variable, but at any node, the two restrictions

$$y_t \leq \lfloor y_t \rfloor \text{ or } y_t \geq \lfloor y_t \rfloor + 1$$

may be imprecise, that is there may not exist any set $O'_t \subseteq O_t$ such that $\sum_{o \in O'_t} h_{ot} = \lfloor y_t \rfloor$ or $\sum_{o \in O'_t} h_{ot} = \lfloor y_t \rfloor + 1$.

To efficiently branch with an ECB constraint of the form (V.3) entails finding the feasible values of y_t : $y'_{t_1}, y'_{t_2}, y'_{t_3}, ..., y'_{t_m}$, perhaps by dynamic programming, and enforce branchings so that if

$$y'_{t_k} < \widehat{y}_t < y'_{t_{k+1}},$$

then the branching rule will be

$$y_t \leq y'_{t_k} \text{ or } y_t \geq y'_{t_{k+1}},$$

where possibly $y'_{t_{k+1}} - y'_{t_k} \neq 1$. It is possible to branch on y_t where the values $y'_{t_1}, y'_{t_2}, y'_{t_3}, ..., y'_{t_m}$ are specified by a sequence with some solvers (e.g., the XA Solver, (*GAMS-The Solver Manual*, 1993)), but we are not guaranteed that the feasible values of y_t will correspond to any given sequence. We will not pursue this constraint branching technique in this dissertation.

2. Nested ECB

When using nested ECB, it is important to remember that our goal is to reduce the amount of work the variable branch-and-bound process must do. If we add too many ECB constraints, we can easily defeat this purpose. For example, we could add an ECB constraint (V.2) and then split J_i into two disjoint subsets J_{i_m} and J_{i_n} and add constraints for them. If we recursively continue this process until we eventually form ECB constraints containing a single binary variable, then we have added many more general integer variables than the number of binary variables that we began with, which will likely be counterproductive. In practice, we have discovered that adding only a few nested ECB constraints for each basic constraint works well. The decision of the number of ECB constraints to add and their composition depends on the procedure that is used.

a. Static ECB

For static nested ECB, we use the number of variables x_j in the basic ECB constraint to determine the number of nested constraints to add to a problem. In general, for each set of variables $J_i \subseteq J$, (where J is the set of variables in the problem), the technique

adds an ECB constraint

$$\sum_{j \in J_i} x_j - y_i = 0, \tag{V.4}$$

where y_i is a general integer variable, and then adds constraints

$$\sum_{j \in J_{i_k}} x_j - y_{i_k} = 0, \; k \in K,$$

where $K = \{1, ..., n\}$, $J_i = \bigcup_{k=1}^{n} J_{i_k}$ and $J_{i_m} \cap J_{i_p} = \emptyset \; \forall \; m, p \in K$. When adding these constraints to a MIP, we set branching priorities such that y_i has the highest priority, the y_{i_k}, $k \in K$, have the next highest priority, and the x_j have the lowest priority.

We apply nested ECB to the GAP by adding the basic ECB constraints

$$\sum_{o \in O_t} x_{ot} - y_t = 0 \; \forall \; t \in T, \tag{V.5}$$

and then adding only one level of nested constraints

$$\sum_{o \in O_{t_k}} x_{ot} - y_{t_k} = 0 \; \forall \; t \in T, \; \forall k \in K_t,$$

where the O_{t_k} are disjoint sets of variables in the nested ECB constraints, $O_{t_k} \subset O_t \; \forall \; k$, K_t is the set of indices of the nested ECB constraints, and $\bigcup_{k=1}^{|K_t|} O_{t_k} = O_t$. For this application, $|K_t|$ depends on $|O_t|$. For our test problems, we have found that if $|O_t|$ is small, say $|O_t| < 8$, it makes little sense to form any nested constraints, and the basic ECB constraint will suffice for that particular knapsack constraint. If $8 \leq |O_t| < 16$, we add $|K_t| = \lfloor |O_t| / 2 \rfloor = 2$ nested ECB constraints, each of which will also have at least four x_{ot} variables. If $|O_t| \geq 16$, we create $|K_t| = \lfloor |O_t| / 2 \rfloor = 4$ nested ECB constraints, each of which will have at least four x_{ot} variables.

When using the technique of nested ECB as described above, there is always a redundant ECB constraint formed. Thus, we omit one of the nested ECB constraints when using this technique.

Lemma 5.1 *When partitioning the variables in a basic ECB constraint into $|K|$ disjoint subsets such that $K = \{1, ..., n\}$, $J_i = \bigcup_{k=1}^{n} J_{i_k}$ and $J_{i_m} \cap J_{i_p} = \emptyset \; \forall \; m, p \in K$, only $n - 1$*

nested ECB constraints need to be added to achieve the branching effect of n nested ECB constraints.

Proof. Consider the basic ECB constraint

$$\sum_{j \in J_i} x_j - y_i = 0. \tag{V.6}$$

In order to add the n nested constraints, we partition the variables $j \in J_i$ into n disjoint subsets such that $K = \{1, ..., n\}$, $J_i = \bigcup_{k=1}^{n} J_{i_k}$ and $J_{i_m} \cap J_{i_p} = \emptyset \; \forall \, m, p \in K$. Now, we form the n constraints of the form

$$\sum_{j \in J_{i_k}} x_j = y_{i_k}.$$

If we add the first $n-1$ nested constraints, when y_i and y_{i_k} are integer for $k = 1, 2, ..., n-1$, the higher branching priority of y_i ensures that

$$\sum_{j \in J_{i_n}} x_j = y_i - y_{i_1} - ... - y_{i_{n-1}}. \quad QED$$

We have discussed one example of adding nested constraints in the context of static ECB. Next, we discuss how we use a semi-dynamic procedure to form nested ECB constraints.

b. Semi-dynamic ECB

Semi-dynamic ECB uses information from one or more LP solutions to determine the composition of the nested ECB constraints. We can implement this technique by simply solving the LP relaxation of the IP, order the fractional variables in some meaningful way (perhaps focusing on those variables with $\widehat{x}_j \approx .5$; i.e., "furthest" from being integer), and apportion the most important fractional variables as evenly as possible among the nested ECB constraints.

When we combine ECB and knapsack constraint generation in composite enumeration, we have information from the LP solutions in the constraint generation phase that we can heuristically use to decide how to implement ECB. In the constraint generation phase, we typically solve an LP, generate knapsack cuts (if they exist) for each knapsack

constraint, solve the modified LP and recursively generate cuts and solve LPs until no more knapsack cuts can be found.

To implement semi-dynamic ECB, we form a vector f, and use it to record the number of LP solutions f_j in which the variable $\widehat{x}_j$ is fractional in the constraint generation phase. We use this information to decide the composition of the subsets of variables for each nested ECB constraint and also use the information to decide how many nested ECB constraints will be added for a particular basic ECB constraint. Intuitively, we hope that variables that are nearly always integer (f_j small) in the constraint generation phase will stay integer in the LP solutions during the branch and bound. This should allow our nested ECB constraints to focus on the variables that are more likely to be fractional in the branch and bound (f_j large).

Just as with the description of SOS Type 1 in Chapter I, we would like to choose subsets of variables so that those with fractional $\widehat{x}_j$ are evenly apportioned among all subsets. If there are $|K_t|$ nested ECB constraints to be added, then the variable that has been fractionated the most is placed in subset J_{i_1}, the next most fractionated variable is placed in J_{i_2}, and so on until the first $|K_t|$ variables are each placed in separate ECB constraints. The $(|K_t|+1)$st variable is then placed in subset J_{i_1}, the $(|K_t|+2)$nd in J_{i_2}, and so on until all variables are placed in a subset, effectively distributing the most-often fractionated variables fairly evenly among the $|K_t|$ nested constraints.

3. Comparison of Dynamic and Static Techniques

After developing the technique of ECB, we discovered a reference in Jörnsten and Värbrand (1991) to a technique called "generalized branching" developed by Jörnsten and Larsson (1988). (The original research report is unavailable at this time.) They describe two different branching techniques for the generalized assignment problem, but the one that resembles ECB is of interest here. Their dynamic technique at a node v_j examines the knapsack constraints

$$\sum_{o \in O_t} h_{ot} x_{ot} \leq H_t,$$

and checks to see if $\sum_{o \in O_t} \widehat{x}_{ot}$ is fractional in the solution to $LP(j)$. If so, the restrictions

derived at v_j are

$$\sum_{o \in O_t} x_{ot} \leq \left\lfloor \sum_{o \in O_t} \widehat{x}_{ot} \right\rfloor \text{ or } \sum_{o \in O_t} x_{ot} \geq \left\lfloor \sum_{o \in O_t} \widehat{x}_{ot} \right\rfloor + 1.$$

Thus, $LP\,(j_1)$ at v_{j_1} is $LP\,(j)$ with the first restriction added, and $LP\,(j_2)$ at v_{j_2} is $LP\,(j)$ with the second restriction added. This means that for each truck t, there is a constraint added each time $\sum_{o \in O_t} x_{ot}$ is fractional and chosen to derive restrictions at a particular node. Static basic ECB in our GAP application described in Section V.2 will branch in exactly the same way, except we have no need to modify the standard variable-based branch and bound. Because we have added a basic ECB constraint for each knapsack constraint in the GAP,

$$\sum_{o \in O_t} x_{ot} - y_t = 0 \; \forall \, t \in T,$$

and set the branching priority higher on the y_t than for the x_{ot}, branch and bound will enforce the same constraints explicitly added by generalized branching by simply enforcing the branching priorities specified by the ECB technique.

VI. COMPUTATIONAL RESULTS

Knapsack cuts and explicit-constraint branching have been implemented and tested on sets of generalized assignment problems (GAPs). Explicit-constraint branching has also been tested on set-partitioning and other problems. We begin by giving an overview of the implementation. Then, we summarize computational results from a set of 84 randomly generated GAPs. (Detailed results of these problems are presented in the Appendix.). We present detailed results of eight real-world GAPs (standard and elastic) from the petroleum industry and several binary integer problems, including some set-partitioning problems.

The composite solution algorithm is coded in C and uses CPLEX 3.0's callable library (CPLEX, 1993) to solve linear programming relaxations and to perform branch and bound on the integer program as modified by our techniques. If knapsack cuts are to be added to an IP, an LP is solved, and a knapsack cut is added for each eligible knapsack constraint if such a cut exists. This procedure is iterative, so if any cut is added in the first iteration, a new LP is solved in a second iteration, cuts are again derived from eligible constraints, if possible, and so on. Cuts added in previous iterations are candidates for cut generation as well since they are also knapsack constraints. This iterative process terminates when no more cuts can be found, or after 20 iterations of adding knapsack cuts, whichever occurs first. Then, if ECB will be used, ECB constraints and variables are added and branching priorities are specified for the general integer branching variables. After all knapsack and ECB constraints are added, the modified IP is then solved by CPLEX's branch-and-bound solver. All CPU times reported here are from an IBM RS-6000 Model 590 with 512 megabytes of random access memory. We allow a maximum of 1000 CPU seconds with a relative optimality criterion of 0.5% for all GAPs. Optimality criteria for other problems are specified elsewhere.

A. GENERALIZED ASSIGNMENT PROBLEM

The GAP is difficult; real-world problems with as few as 200 variables cannot be solved with standard branch and bound. We use two sets of GAPs to test knapsack cuts and

basic ECB. The first set consists of 84 randomly generated problems taken from the literature (Osman, 1994, Beasley and Chu, 1995, Cattrysse et al., 1994, Beasley, 1997). These problems allow each order to be delivered by each truck. The second set of eight real-world problems (Brown, 1995) has many orders that can only be delivered by specific trucks; this apparently makes these problems more difficult to solve than randomly generated GAPs of comparable size.

The basic formulation of the GAP (in terms of delivering orders on trucks) is

$$\begin{aligned} \text{minimize} \quad & \sum_{o \in O} \sum_{t \in T} c_{ot} x_{ot} & & \\ \text{subject to:} \quad & \sum_{t \in T_o} x_{ot} & = & \; 1 \; \forall o \; (ORDERS) \\ & \sum_{o \in O_t} h_{ot} x_{ot} & \leq & \; H_t \; \forall t \; (TRUCKHOURS) \\ & x_{ot} & \in & \; \{0,1\} \; \forall o \in O_t, \; t \in T_o. \end{aligned}$$

(A complete description of the GAP with all data, indices, and variables defined appears in Chapter II.) To implement explicit-constraint branching, the GAP is modified to

$$\begin{aligned} \text{minimize} \quad & \sum_{o \in O} \sum_{t \in T} c_{ot} x_{ot} & & & \\ \text{subject to:} \quad & \sum_{t \in T_o} x_{ot} & & = & \; 1 \; \forall o \; (ORDERS) \\ & \sum_{o \in O_t} h_{ot} x_{ot} & & \leq & \; H_t \; \forall t \; (TRUCKHOURS) \\ & \sum_{o \in O_t} x_{ot} & - y_t & = & \; 0 \; \forall t \\ & x_{ot} & & \in & \; \{0,1\} \; \forall o \in O_t, \; t \in T_o \\ & & y_t & \in & \; \{0,1,...,|O_t|\} \; \forall t. \end{aligned}$$

B. RESULTS FOR RANDOMLY GENERATED GAPS

We summarize the problem size and results for the 84 randomly generated GAPs in Tables 2, 3 and 4; detailed results appear in the Appendix. These problems are solved with a maximizing objective function. The notation used in the tables of results is:

BandB. Branch-and-bound solver only. We attempt to solve these problems using only the CPLEX branch-and-bound solver, branching on the variable with maximum infeasibility. (This branching option was used throughout all computational tests.)

# Variables	Trucks	Orders	Constraints	Variables
<1000	7	40	47	300
≥1000	13	160	173	2000

Table 2. Average problem size for randomly generated GAPs. There are 64 problems with under 1000 variables, and 20 that have at least 1000 variables.

KS. Knapsack cuts are added before applying the CPLEX branch and bound solver to the modified IP.

ECB. ECB constraints and variables are added and branching priorities specified before applying the CPLEX branch and bound solver to the modified IP.

KS/ECB. Knapsack cuts and ECB constraints and variables are added and branching priorities specified before applying the CPLEX branch and bound solver to the modified IP.

Solved. Percentage of the problems that were successfully solved in under 1000 CPU seconds.

CPU secs. The average time in seconds required to solve each problem that could be solved in under 1000 CPU seconds.

Nodes. The average number of nodes in the branch-and-bound tree required by each problem that was solved in under 1000 CPU seconds. The letter "K" indicates thousands of nodes.

	BandB	KS	ECB	KS/ECB
Solved	40.6%	100%	100%	100%
CPU secs	88.2	4.3	6.9	2.9
Nodes	33K	451	1022	33

Table 3. Summary of computational results for random GAPs with fewer than 1000 variables. An average of 22 knapsack cuts are added per problem for each solution procedure using knapsack cuts (KS and KS/ECB). The average time and nodes for branch and bound reflects only those problems that were solved in under 1000 CPU seconds.

As seen in Table 3, over half of the smaller GAPs cannot be solved by standard branch and bound alone, but any of the techniques developed in this dissertation will solve all 64 of these problems. Table 4, which covers the large problems, also demonstrates the inability of standard branch and bound to solve GAPs. More importantly, it demonstrates that some of the larger GAPs cannot be solved when knapsack cuts or ECB are applied individually, but used together, these techniques solve all of the randomly generated GAPs.

	BandB	KS	ECB	KS/ECB
Solved	30%	80%	90%	100%
CPU secs	352.5	113.0	26.5	70.1
Nodes	49K	3731	1596	724

Table 4. Averaged results for random GAPs with 1000 variables or more. An average of 42 knapsack cuts were added for each problem successfully solved with knapsack cuts and branch and bound, and an average of 48 knapsack cuts were added for each problem solved with knapsack cuts, ECB and branch and bound. "CPU secs" and "Nodes" reflect average data for those problems solved in under 1000 CPU seconds.

C. RESULTS FOR REAL-WORLD GAPS

Here we test ECB and knapsack cuts on a set of eight real-world GAPS from the petroleum industry. Table 5 lists the basic statistics for these problems.

Model Name	Trucks	Orders	Constraints	Variables
LONGD	8	22	30	46
LONGN	6	21	27	37
BOSTD	17	56	73	330
BOSTN	15	50	65	266
DLWRD	19	70	89	469
DLWRN	11	48	59	200
LOSAD	34	151	185	1835
LOSAN	35	147	182	1790

Table 5. Problem statistics for real-world GAPs.

Table 6 lists solution times and the number of branch-and-bound nodes for all solution techniques applied to these GAPS. However, four of the problems are infeasible (denoted "INF" in the "Time" column) because not all orders can be delivered in the time allotted to the trucks. These problems must be "elasticized" in order to obtain feasibility. We address solving the elastic version of these GAPs later in this chapter.

Ignoring the infeasible problems, Table 6 shows that knapsack cuts solve only one real-world problem when used alone. This stands in sharp contrast with the randomly generated GAPs. ECB solves all problems with the exception of DLWRD. As with the randomly generated GAPs, all (feasible) problems are solved when ECB and knapsack cuts are combined.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
LONGD	INF	N/A	INF	N/A	N/A	INF	N/A	INF	N/A
LONGN	INF	N/A	INF	N/A	N/A	INF	N/A	INF	N/A
BOSTD	(1000)	(151K)	(1000)	(138K)	41	11.10	2234	31.62	4439
BOSTN	(1000)	(158K)	(1000)	(133K)	44	11.47	2920	3.89	255
DLWRD	(1000)	(119K)	(1000)	(111K)	57	(1000)	(119K)	19.22	2187
DLWRN	(1000)	(146K)	5.82	1050	40	1.66	60	3.28	109
LOSAD	INF	N/A	INF	N/A	N/A	INF	N/A	INF	N/A
LOSAN	INF	N/A	INF	N/A	N/A	INF	N/A	INF	N/A

Table 6. GAP solution results. Time is in CPU seconds. Problems with CPU time "(1000)" could not be solved in 1000 CPU seconds. Nodes are the number of nodes in the branch and bound enumeration tree ("K" indicates thousands). The number of knapsack cuts added is the same for KS and KS/ECB solutions, so the information is only recorded once. Problems with "INF" in the "Time" column are infeasible in their present form, and require elastic constraints to insure feasiblity.

D. RESULTS FOR REAL-WORLD, ELASTIC GAPS

Here we investigate computation involving elastic GAPs. In particular, we modify the eight real-world GAPs by changing the knapsack constraints to allow penalized violation. This can be thought of as allowing the trucks to operate in an overtime status with the penalty p_t^+ being the cost of the overtime for truck t. The four real-world problems that were infeasible can be solved with the addition of these elastic variables. The elastic formulation of the GAP with ECB constraints added is:

$$
\begin{array}{lrcll}
\text{minimize} & \sum_{o\in O}\sum_{t\in T} c_{ot}x_{ot} + \sum_{t\in T} p_t^+ z_t^+ & & & \\
\text{subject to:} & \sum_{t\in T_o} x_{ot} & = & 1 & \forall\, o \in O \\
& \sum_{o\in O_t} h_{ot}x_{ot} - z_t^+ & \leq & H_t & \forall\, t \in T \\
& \sum_{o\in O_t} x_{ot} - y_t & = & 0 & \forall\, t \in T \\
& x_{ot} & \in & \{0,1\} & \forall\, o \in O, \\
& z_t^+ & \in & \{0,1,...,H_t^\delta\} & \forall\, t \in T \\
& y_t & \in & \{0,1,...,|O_t|\}\ \forall t. & \forall t \in T
\end{array}
$$

Table 7 shows solution times and the number of branch and bound nodes when each technique is applied to the elasticized GAP. Interestingly, the table shows that the problems that were feasible and solved without the benefit of elastic constraints (compare with Table

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
LONGD	1.69	259	2.19	6	5	1.78	4	2.20	12
LONGN	1.53	82	2.11	95	4	1.48	12	2.11	4
BOSTD	(1000)	(170K)	(1000)	(129K)	46	2.50	159	3.52	63
BOSTN	(1000)	(183K)	(1000)	(132K)	35	1.75	57	3.00	50
DLWRD	(1000)	(113K)	(1000)	(102K)	60	(1000)	(137K)	8.61	492
DLWRN	28.92	10441	7.99	1393	34	1.75	89	3.10	65
LOSAD	(1000)	(79K)	(1000)	(49K)	117	(1000)	(35K)	199.03	3725
LOSAN	(1000)	(73K)	(1000)	(34K)	181	(1000)	(36K)	815.94	13627

Table 7. Elastic GAP solution results. Time is in CPU seconds. Nodes are the number of nodes in the branch and bound enumeration tree. The number of knapsack cuts added is the same for KS and KS/ECB solutions, so the information is only recorded once.

6) were all solved more quickly and with fewer nodes after being elasticized. More importantly, problems that were infeasible are now solved. This table, like Table 4, reflects our general experience: The combination of ECB and knapsack cuts solves more problems than either technique applied in isolation.

E. GAP RESULTS USING NESTED ECB

1. Nested ECB Without Knapsack Cuts

We first test nested ECB on problems that cannot be solved by basic ECB, or problems that need more than 10000 branch and bound nodes to solve using basic ECB. "Nest n" refers to the addition of n nested ECB constraints, one of which is implicit. "SD" refers to semi-dynamic nesting which apportions variables to nested constraints using LP solution information. In this case, the apportionment is accomplished by solving one LP and applying the distance metric described in Chapter V to LP solution values. "Static" implies that nesting was carried out by arbitrarily splitting the variables into the appropriate number of disjoint subsets, all of roughly the same size. All problems with "GAP" in the model name are from the set of randomly generated GAPs.

The results, shown in Table 8, are varied. GAP11c is solved best by nesting four semi-dynamic constraints. GAP11e solves more quickly using two static nested constraints.

Model Name	Var	Static Nest 2		Static Nest 4		SD Nest 2		SD Nest 4	
		Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
GAP11c	500	98.63	23824	6.22	1093	28.10	7397	5.29	1015
GAP11e	500	39.84	9601	233.62	53070	123.44	28546	75.85	16942
GAPC3*	1000	(1000)	(136K)	253.59	30443	(1000)	(133K)	(1000)	(132K)
GAPA5*	2000	(1000)	(87K)	(1000)	(92K)	(1000)	(84K)	(1000)	(98K)
GAPC5	2000	(1000)	(81K)	(1000)	(92K)	(1000)	(78K)	(1000)	(87K)
DLWRD*	469	(1000)	(120K)	(1000)	(118K)	(1000)	(118K)	(1000)	(111K)

Table 8. Nested ECB without knapsack cuts. Time is in CPU seconds. Nodes are the number of nodes in the branch and bound enumeration tree. "Var" indicates the number of variables in the problem. An asterisk (*) indicates that the problem could not be solved by basic ECB.

More nesting is not always better as GAP11e demonstrates. GAPA5 and DLWRD could not be solved by any type of ECB technique, and although GAPC3 was solved by nesting four constraints, the basic ECB solution was better. Note that the DLWRD problem (the only real-world problem) has the smallest number of variables, yet cannot be solved by any of the ECB techniques.

2. Nested ECB With Knapsack Cuts

We also test nested ECB combined with knapsack cuts. We select five problems that require over 1000 nodes to solve using basic ECB and knapsack cuts. The semi-dynamic apportionment is accomplished by recording the number of LP solutions (all solutions obtained during the cut-generation phase) in which a particular variable was fractional, and apportioning variables to the nested constraints based on the "propensity" of the variable to be fractional.

Model Name	Var	Static Nest 2		Static Nest 4		SD Nest 2		SD Nest 4	
		Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
GAPC3	1000	17.15	1388	31.77	2968	13.50	992	30.19	3374
GAPA5	2000	(1000)	(50K)	(1000)	(70K)	581.79	25980	(1000)	(69K)
GAPB5	2000	7.14	57	175.18	6388	16.2	392	137.51	5434
BOSTD	330	19.18	2536	48.73	5644	29.3	3619	21.10	2276
DLWRD	469	150.91	15523	509.67	47680	876.44	83274	(1000)	(87K)

Table 9. Nested ECB with knapsack cuts. Time is in CPU seconds. Nodes are the number of nodes in the branch and bound enumeration tree.

The results, shown in Table 9, are again varied. GAPA5 and DLWRD do not benefit from nesting having better solution times with knapsack cuts and basic ECB. Semi-dynamic nesting with two nested constraints solves GAPC3 most quickly, while semi-dynamic ECB with four and static ECB with two nested constraints both solve BOSTD in about 20 CPU seconds, quicker than any other solution technique. GAPB5 solves quickly with both the static and semi-dynamic techniques using two nested constraints. In conclusion, we can say that nested ECB may be useful for solving certain problems, and sometimes the static technique will be better and sometimes the semi-dynamic technique will be. Computational testing will be necessary to determine if nesting is of value for a given set of problems.

F. STRUCTURE-INDEPENDENT RESULTS WITH ECB

Here we test structure-independent ECB on set-partitioning problems and other problems (MIPs) containing binary variables to demonstrate that special structure is not required to use ECB successfully.

1. Set-Partitioning Problems and General MIPs

Set-partitioning problems (SPPs) are difficult IPs to solve and tend to be much larger than GAPs. The basic formulation of the SPP is

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in J} c_j x_j & & \\ \text{subject to:} \quad & \sum_{j \in J} a_{ij} x_j & = & \ 1 \ \forall \ i = 1, ..., m \\ & x_j & \in & \ \{0,1\} \ \forall \ j \in J, \end{aligned}$$

where $a_{ij} \in \{0,1\}$. We implement ECB by using just one ECB constraint that ensures that the sum of all the variables in the problem is integer before branching on any individual

binary variable. The modified model is:

$$\begin{array}{lrcl} \text{minimize} & \sum_{j \in J} c_j x_j & & \\ \text{subject to:} & \sum_{j \in J} a_{ij} x_j & = & 1 \; \forall \, i = 1, ..., m \\ & \sum_{j \in J} x_j \; - y & = & 0 \\ & x_j & \in & \{0, 1\} \; \forall \, j \in J \\ & y & \in & \{1, ..., m\}. \end{array}$$

A MIP can contain continuous, general integer and binary variables. We also test ECB on more general MIPs (than the SPP) by adding a single ECB constraint of the form

$$\begin{array}{rcl} \sum_{j \in J} x_j \; - y & = & 0 \\ y & \in & \{0, 1, ..., |J|\}. \end{array}$$

Here, J is the index set for the binary variables in a problem and y is the added general integer branching variable.

2. ECB Results

To test structure-independent ECB, we obtained 12 integer problems from the MIPLIB at Rice University (Bixby, et al. 1997) and two SPPs (SETP1 and PIB00) from the meat-packing industry (Brown, 1997). We decided to use MIPLIB problems that branch and bound alone could solve in a reasonable time to a reasonable optimality criterion. We allowed 10000 CPU seconds, and solved to an optimality criterion of 0.5%. Results are shown in Table 10.

Model Name	Total Vars	Binary Vars	Gen Int Vars	BandB		ECB	
				Time	Nodes	Time	Nodes
SETP1†	2564	2564	0	68.4	1608	7.5	37
SETP2	2564	2564	0	23.2	373	8.6	121
PIB00	377	377	0	.6	39	.6	4
MOD008†	319	319	0	14.1	5639	1.6	405
CAP6000†	6000	6000	0	1983.4	4654	120.1	776
MISC03	160	159	0	8.8	945	14.1	1270
L152LAV†	1989	1989	0	151.7	3231	156.9	3381
MISC06	1808	112	0	2.4	20	5.4	53
ENIGMA	100	100	0	1.5	337	1.6	331
LSEU	89	89	0	52.9	25089	61.3	24721
P0033	33	33	0	6.0	5806	7.6	6081
P0201	201	201	0	7.7	855	11.8	1375
DCMULTI	548	75	0	12.5	1134	30.6	1429
RGN	180	100	0	15.1	5275	15.1	5232
BELL3A	133	39	32	13.2	4516	6.8	2317

Table 10. Solving binary integer problems with structure-independent ECB. Time is in CPU seconds. Nodes are the number of nodes in the branch and bound enumeration tree. "Var" indicates the number of variables in the problem. SETP2 is a modification of SETP1 with adjusted penalties for constraint-violation variables. For all those problems marked with "†," the CPLEX maximum-infeasibility branching strategy was used; that strategy was better than the default. Otherwise, the CPLEX default option was used which allows CPLEX to determine the branching strategy.

ECB helped significantly with SETP1, SETP2, PIB00, MOD008, CAP6000 and BELL3A. L152LAV, ENIGMA, LSEU, P0033 and RGN solved in about the same time and number of nodes by both methods. Branch and bound was the better solution option for the other four problems.

VII. SUMMARY AND RECOMMENDATIONS

This dissertation has developed enhanced composite enumeration techniques for solving IPs and MIPs. Results demonstrate that previously unsolvable problems can be solved with these new techniques. This dissertation has specifically examined constraint generation from the knapsack polytope and "explicit-constraint branching," a technique that applies constraint branching to IPs and MIPs that do not have the special problem structure required by "implicit-constraint branching."

A. CONTRIBUTIONS

1. Knapsack Cuts

In Chapter III, we developed knapsack cut-finding procedures for minimal cover cuts, and converted existing cut-strengthening theory into practical procedures that lift and tighten violated minimal cover valid inequalities to violated knapsack facets in polynomial time. We defined a new class of knapsack cuts called "non-minimal cover cuts," cuts that do not contain a violated minimal cover cut, developed a dynamic programming procedure to find them, and developed a new method, "deficit lifting," to lift them. Deficit lifting enables all non-minimal cover cuts to be lifted and tightened to violated facets as well.

The minimal cover cut-finding procedure finds a maximally violated minimal cover cut, if one exists, in pseudo-polynomial time using dynamic programming. All violated minimal cover inequalities are lifted to strong cover cuts by a new procedure called "interior lifting." The procedure that tightens strong cover cuts to violated facets improves upon the polynomial-time facet-finding algorithm of Zemel (1989).

Extensions to knapsack cuts were developed for elastic knapsack constraints and non-standard knapsack constraints (those with greater-than-or-equal-to and equality senses). Elastic knapsack cuts are generated by an adaptation of the minimal cover cut-finding problem of Chapter III. All the procedures of Chapter III were adapted for use with greater-than-

or-equal-to and equality knapsack constraints to allow the generation of facets or at least strong cuts for these types of constraints.

2. Explicit-Constraint Branching

The method of "explicit-constraint branching" was developed in Chapter V. This technique applies constraint branching to MIPs lacking the special structure required by standard "implicit-constraint branching" techniques. Explicit-constraint branching alone solves some MIPs that branch-and-bound cannot, and nesting the ECB constraints often solves more difficult problems. When combined with knapsack cuts, the synergistic effect of ECB with knapsack cuts solves still more difficult IPs.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

This dissertation developed the new technique of explicit-constraint branching. Further study on nesting strategies and semi-dynamic techniques is necessary. ECB may prove to be effective (or not) for other classes of problems, and there may be different adaptations of ECB that could also prove effective for solving difficult problems.

Finding violated knapsack facets that are not associated with any minimal or non-minimal cover cut is an area that requires further study. There could exist a relatively simple dynamic programming technique similar to our minimal cover cut-finding procedure that can find these violated facets.

The implementation of "branch and cut" using easily generated, globally valid knapsack cuts is warranted. In particular, globally valid knapsack cuts that were not apparent from the initial LP relaxations may become apparent and could be useful in solving the restricted problems of the branch and bound. This dynamic constraint generation technique could prove to be effective for solving problems such as the generalized assignment problem.

This dissertation has studied cuts based on special structure. We suggest further study of structure-independent cutting-plane techniques such as the Chvátal-Gomory rounding method (Chvátal,1973). These cuts have been shown to be useful, but our experience

has shown that the methodology can also be computationally burdensome. Since many of these techniques have finite (convergent) algorithms associated with them, the development of computationally efficient, structure-independent cutting planes should be explored further.

LIST OF REFERENCES

Aboudi, R. Hallefjord, A. and Jörnsten, K. (1991), "A facet generation and relaxation technique applied to an assignment problem with side constraints," *European Journal of Operations Research* **50**, 335-344.

Amini, M. and Racer, M. (1994), "A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem," *Management Science* **40**, 868-890.

Balas, E. (1971), "Intersection Cuts-A New type of Cutting Planes For Integer Programming," *Operations Research* **19**, 19-39.

Balas, E. (1975), "Facets of the Knapsack Polytope," *Mathematical Programming* **8**, 146-164.

Balas, E. (1977), "Some valid inequalities for the set partitioning problem," in *Studies in Integer Programming* (P. L. Hammer, et al., eds.) *Annals of Discrete Mathematics* **1**, 13-47.

Balas, E. (1979), "Disjunctive Programming," in *Studies in Integer Programming* (P. L. Hammer, et al., eds.) *Annals of Discrete Mathematics* **5**, 3-51.

Balas, E., Bowman, V. J., Glover, F., and Sommer, D. (1971), "An Intersection Cut From the Dual of a Unit Hypercube," *Operations Research* **19**, 40-44.

Balas, E., Ceria, S., and Cornuéjols, G. (1993), "A Lift-and-Project Cutting Plane Algorithm for Mixed 0-1 Programs," *Mathematical Programming,* **58**, 295-324.

Balas, E., Ceria, S., and Cornuéjols, G. (1996), "Mixed 0-1 Programmming by Lift-and-Project in a Branch and Cut Framework," *Management Science,* **42**, 1229-1246.

Balas, E. and Zemel, E. (1977), "Critical Cutsets of Graphs and Canonical Facets of Set-packing Polytopes," *Mathematics of Operations Research* **2**, 15-19.

Balas, E. and Zemel, E. (1978), "Facets of the Knapsack Polytope from Minimal Covers," *SIAM Journal of Applied Mathematics* **34**, 119-148.

Balas, E. and Zemel, E. (1980), "An Algorithm for Large Zero-One Knapsack Problems," *Operations Research* **28**, 1130-1154.

Balas, E. and Zemel, E. (1984), "Lifting and complementing yields all the facets of positive zero-one programming polytopes," in *Proceedings of the International Conference on Mathematical Programming,* (R. W. Cottle, et al., eds.), Elsevier, North-Holland, Amsterdam, 13-24.

Barnhart, C., Johnson, E., Nemhauser, G. and Vance, P. (1994), "Solving Binary Cutting Stock Porblems by Column Generation and Branch-and-Bound," *Computational Optimization and Applications* **3**, 111-130.

Beale, E. M. L. and Tomlin, J. A. (1970), "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables," in *Proceedings of the 5th International Operations Research Conference,* J. R. Lawrence (ed.), Tavistock Publications Limited, London, 447-454.

Beasley, J. E. "Generalised Assignment Problem," *OR-Library*, The Management School at Imperial College of Science, Technology, and Medicine, London, United Kingdom, available from http://mscmga.ms.ic.ac.uk/, Internet, accessed 10 February 1997.
P. C. Chu and J. E.

Beasley, J. E. and Chu, P. C. (1995), "A genetic algorithm for the generalised assignment problem,"
working paper, The Management School, Imperial College, London SW7 2AZ, England.

Bixby, R. E. , Ceria, S., McZeal, C. M. and Savelsbergh, M. W. P. *MIPLIB 3.0*, Computational and Applied Mathematics Department, Rice University, Houston, Texas, available from http://www.caam.rice.edu/~bixby/miplib/miplib.html, Internet, accessed 10 February 1997.

Boyd, E. A. (1990), "The Lagrangian and other Primal Cutting Planes for Linear Integer Programming Problems," TR90-3, Department of Mathematical Sciences, Rice University, Houston, Texas.

Boyd, E. A. (1990), "Generating Fenchel Cutting Planes for Knapsack Polyhedra," TR90-20, Department of Mathematical Sciences, Rice University, Houston, Texas.

Boyd, E. A. (1992), "A Pseudopolynomial Network Flow Formulation for Exact Knapsack Separation," *Networks* **22**, 503-514.

Boyd, E. A. (1994), "Fenchel Cutting Planes for Integer Programming," *Operations Research* **42**, 53-64.

Bixby, R. E. and Lee, E. K. (1993), "Solving a Truck Dispatching Scheduling Problem Using Branch-and-Cut," TR93-37, Department of Computational and Applied Mathematics, Rice University, Houston, Texas.

Brown, G., personal communication, October 14, 1995.

Brown, G., personal communication, May 9, 1997.

Brown, G. and Graves, G. (1981), "Real-Time Dispatch of Petroleum Tank Trucks," *Management Science* **27**, 19-32.

Brown, G., Graves, G. and Honczarenko, M. (1987), "Design and Operation of a Multicommodity Production/Distribution System Using Primal Goal Decomposition," *Management Science* **33**, 1469-1480.

Caprara, A., and Fischetti, M. (1996), "{0,1/2}-Chvatal-Gomory cuts," *Mathematical Programming* **74**, 221-235.

D. Cattrysse, D., Salomon, M. and Van Wassenhove, L. N. (1994), "A Set Partitioning Heuristic for the Generalized Assignment Problem," *European Journal of Operational Research*, **72**, 167-174.

Chvátal, V. (1973), "Edmonds Polytopes and a Hierarchy of Combinatorial Problems," *Discrete Mathematics* **4**, 305-337.

Chvátal, V. (1975), "On Certain Polytopes Associated with Graphs," *Journal of Combinatorial Theory (B)* **18**, 138-154.

Chvátal, V. (1985), "Cutting Planes in Combinatorics," *European Journal of Combinatorics* **6**, 217-226.

Chvátal, V. and Hammer, P. L. (1977), "Aggregation of Inequalities in Integer Programming," in *Studies in Integer Programming* (P. L. Hammer, et al., eds.) *Annals of Discrete Mathematics* **1**, 145-162.

Cook, W., Rutherford, T., Scarf, H., and Shallcross, D. (1993), "An Implementation of the Generalized Basis Reduction Algorithm for Integer Programming," *ORSA Journal on Computing* **5**, 206-212.

Cornuéjols, G., and Sassano, A. (1989), "On the 0,1 Facets of the Set Covering Polytope," *Mathematical Programming* **43**, 45-55.

CPLEX Manual, *Using the CPLEX^{TM} Callable Library and CPLEX^{TM} Mixed Integer Library,* CPLEX Optimization, Inc., Incline Village, Nevada, 1993.

Crowder, H., Johnson, E. L., and Padberg, M. (1983), "Solving Large-Scale Zero-One Linear Programming Problems," *Operations Research* **31**, 803-834.

Dantzig, G. B., "Note on Solving Linear Programs in Integers," *Naval Research Logistic Quarterly,* **6**, 75-76.

Dantzig, G. B., "Linear Programming," *History of Mathematical Programming,* Lenstra, J., Rinnoy Kan, A. and Schrijver, A. (eds.), Elsevier, North-Holland, Amsterdam, 1991, 19-31.

Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M.(1954), "Solution of a Large Scale Traveling Salesman Problem," *Operations Research* **2**, 393-410.

Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M.(1959), "On a Linear Programming, Combinatorial Approach to the Traveling Salesman Problem," *Operations Research,* 7 58-66.

De Simone, C. (1990), "Lifting Facets of the Cut Polytope," *Operations Research Letters* **9**, 341-344.

Dietrich, B. L. and Escudero, L. F. (1992), "On Tightening Cover Induced Inequalities," *European Journal of Operational Research* **60**, 335-343.

Dietrich, B. L. and Escudero, L. F. (1994), "Obtaining Clique, Cover, and Coefficient Reduction Inequalities as Chvatal-Gomory Inequalities and Gomory Fractional Cuts," *European Journal of Operational Research* **73**, 539-546.

Dietrich, B. L., Escudero, L. F. and Chance, F.(1993), "Efficient Reformulation for 0-1 Programs-Methods and Computational Results," *Discrete Applied Mathematics* **42**, 147-175.

Doherty, M. E., Matin, R. K. and Sweeney, D. J. (1985), "The Reduced Cost Branch and Bound Algorithm for Mixed Integer Programming," *Computers & Operations Research* **12**, 139-149.

Faulkner, J. C. and Ryan, D. M. (1988), "On the Integer Properties of Scheduling Set Partitioning Models," *European Journal of Operations Research* **35**, 442-456.

Foster, B. A. and Ryan, D. M. (1981), "An Integer Programming Approach to Scheduling," in *Computer Scheduling of Public Transport* (A. Wren, ed.) North-Holland Publishing Company, 269-280.

GAMS-The Solver Manual, GAMS Development Corporation, Washington, DC, 1993.

Garfinkel, R., and Nemhauser, G. (1972), *Integer Programming,* John Wiley and Sons, New York.

Geoffrion, A. M. and Graves, G. W. (1974), "Multicommodity Distribution System Design by Benders Decomposition," *Management Science,* **20**, 822-844.

Gomory, R. E. (1958), "Outline of an Algorithm for Integer Solutions to Linear Problems," *Bulletin of the American Mathematical Society,* **64**, 275-278.

Gomory, R. E. (1963), "An All-Integer Programming Algorithm," in *Industrial Scheduling,* J. F. Muth, G. L. Thompson (eds.), Prentice-Hall, Englewood Cliffs, 193-206.

Gomory, R. E., "Early Integer Programming," *History of Mathematical Programming,* Lenstra, J., Rinnoy Kan, A. and Schrijver, A. (eds.), Elsevier, North-Holland, Amsterdam, 1991, 55-61.

Gottlieb, E. S. and Rao, M. R. (1990), "The Generalized Assignment Problem: Valid Inequalities and Facets," *Mathematical Programming,* **46**, 31-52.

Gottlieb, E. S. and Rao, M. R. (1990), "$(1,k)$-Configuration Facets for the Generalized Assignment Problem," *Mathematical Programming,* **46**, 53-60.

Guignard, M. and Spielberg, K. (1977), "Propagation,Penalty Improvement and Use of Logical Inequalities," *Methods of Operations Research,* **25,** 157-171.

Hammer, P. L., Johnson, E. L., and Peled, U. N. (1975), "Facet of Regular 0-1 Polytopes," *Mathematical Programming,* **8**, 179-206.

Hartvigsen, D. and Zemel, E. (1992), "The Complexity of Lifted Inequalities for the Knapsack Problem," *Discrete Applied Mathematics* **39**, 113-123.

Hoffman, K. L. and Padberg, M. (1991), "Improving LP-Representations of Zero-One Linear Programs for Branch and Cut," *ORSA Journal on Computing,* **3**, 121-134.

Hoffman, K. L. and Padberg, M. (1993), "Solving Airline Crew Scheduling Problems by Branch and Cut," *Management Science,* **39**, 657-682.

Hummeltenberg, W. (1984), "Implementations of Special Ordered Sets in MP software," *European Journal of Operations Research* **17**, 1-15.

Johnson, E., Kostreva, M. and Suhl, U. (1985), "Solving 0-1 Integer Programming Problems Arising from Large-Scale Planning Models," *Operations Research,* **33**, 803-819.

Jörnsten, K. O. and Larsson, T. (1988), "A Generalized Branching Technique," Department of Mathematics Research Report, Linkoping Institute of Technology.

Jörnsten, K. O. and Värbrand, P. (1991), "A Hybrid Algorithm for the Generalized Assignment Problem," *Optimization,* **22**, 273-282.

Jörnsten, K. O. and Värbrand, P. (1990), "Relaxation Techniques and Valid Inequalities Applied to the Generalized Assignment Problem," *Asia-Pacific Journal of Operations Research,* **7**, 172-189.

Land, A., and Doig, A. (1960), "An Automatic Method for Solving Discrete Programming Problems," *Econometrica, **28,*** 497-520.

Matin, R. K. and Sweeney, D. J. (1983), "An Ideal Column Algorithm for Integer Programs with Special Ordered Sets of Variables," *Mathematical Programming* **26**, 48-63.

Nemhauser, G. L. and Vance, P. H. (1994), "Lifted Cover Facets of the 0-1 Knapsack Polytope with GUB constraints," *Operations Research Letters* **16**, 255-263.

Nemhauser, G. L. and Wolsey, L. A. (1988), *Integer and Combinatorial Optimization,* John Wiley and Sons, New York.

Osman, I. H. (1995), "Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches," *OR Spektrum*, **17**, 211-225.

Padberg, M. (1973), "On the Facial Structure of Set Packing Polyhedra," *Mathematical Programming,* **5**, 199-215.

Padberg, M. (1975), "A Note on Zero-One Programming," *Operations Research,* **23**, 833-837.

Padberg, M. and Rinaldi, G. (1991), "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems," *SIAM Review,* **33**, 60-100.

Parker, R., and Rardin, R. (1988), *Discrete Optimization,* Academic Press, Inc. San Diego.

Peled, U. N. (1977), "Properties of Facets of Binary Polytopes," in *Studies in Integer Programming* (P. L. Hammer, et al., eds.) *Annals of Discrete Mathematics* **1**, 435-456.

Ross, G. T. and Soland, R. M. (1975), "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Mathematical Programming,* **8**, 91-103.

Sassano, A. (1989), "On the Facial Structure of the Set Covering Polytope," *Mathematical Programming,* **44**, 181-202.

Shmoys, D. B. and Tardos, E. (1993), "An Approximation Algorithm for the Generalized Assignment Problem," *Mathematical Programming,* **62**, 461-474.

Sol, M. (1994), "Column Generation Techniques for Pickup and Delivery Problems," Ph.D. thesis, Technische Universiteit Eindhoven.

Van Roy, T. and Wolsey, L. A. (1987), "Solving Mixed Integer Programming Problems Using Automatic Reformulation," *Operations Research,* **35**, 45-57.

Wood, R. K. "Solving a Generalized Generalized Assignment Problem," *INFORMS* presentation, 25 April 1995.

Wilson, J. M. (1990), "Generating Cuts in Integer Programming with Families of Special Ordered Sets," *European Journal of Operations Research* **46**, 101-108.

Wolsey, L. A. (1975), "Faces for a Linear Inequality in 0-1 Variables," *Mathematical Programming,* **8**, 165-178.

Zemel, E. (1978), "Lifting the Facets of Zero-One Polytopes," *Mathematical Programming,* **15**, 268-277.

Zemel, E. (1979), "Easily Computable Facets of the Knapsack Polytope," *Mathematics of Operations Research,* **14**, 760-764.

APPENDIX. COMPUTATIONAL RESULTS

This appendix contains detailed results of the randomly generated generalized assignment problem. The composite solution algorithm is coded in C and uses CPLEX 3.0's callable library to solve linear programming relaxations and to branch and bound on the integer program as modified by our solution algorithm. All CPU times reported here are from an IBM RS-6000 Model 590 with 512 megabytes of random access memory. We allowed a maximum of 1000 CPU seconds with optimality criterion of .5%.

A. STATISTICS

The following table provides probelm size information for the 84 GAP problems.

Model Name	Trucks	Orders	Constraints	Variables
GAP1a-e	5	15	20	75
GAP2a-e	5	20	25	100
GAP3a-e	5	25	30	125
GAP4a-e	5	30	35	150
GAP5a-e	8	24	32	182
GAP6a-e	8	32	40	256
GAP7a-e	8	40	48	320
GAP8a-e	8	48	56	384
GAP9a-e	10	30	40	300
GAP10a-e	10	40	50	400
GAP11a-e	10	50	60	500
GAP12a-e	10	60	70	600
GAPA1-D1	5	100	105	500
GAPA2-D2	5	200	205	1000
GAPA3-D3	10	100	110	1000
GAPA4-D4	10	200	210	2000
GAPA5-D5	20	100	120	2000
GAPA6-D6	20	200	220	4000

Table 11. GAP problem size. The first 60 GAP problems have been used by I. H. Osman, "Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches," OR Spektrum, Volume 17, 211-225, (1995), and D. Cattrysse, M. Salomon and L. N. Van Wassenhove, "A set partitioning heuristic for the generalized assignmnet problem," European Journal of Operational Research, Volume 72, 167-174, (1994). The final 24 problems (GAPA1-D6) were used by P. C. Chu and J. E. Beasley, "A genetic algorithm for the generalised assignment problem," working paper, The Management School, Imperial College, London SW7 2AZ, England (1995).

B. RESULTS FOR PROBLEMS WITH LESS THAN 500 VARIABLES

The first 50 problems presented no challenge for any of the techniques developed in the dissertation, to include using only knapsack cuts in conjunction with the branch and bound. Note that branch and bound alone could not solve over half of the 50 problems.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAP1a	2.19	469	1.98	24	16	1.57	59	1.96	6
GAP1b	12.33	6400	1.97	15	18	1.88	216	2.07	17
GAP1c	2.81	929	1.97	2	17	1.58	62	2.02	3
GAP1d	2.40	588	1.90	6	13	1.67	104	1.97	15
GAP1e	1.73	159	1.97	23	11	1.51	15	1.94	2
GAP2a	40.95	33168	1.96	4	18	1.79	238	2.03	4
GAP2b	4.12	2897	2.02	7	19	1.58	82	2.04	7
GAP2c	14.05	11483	2.26	320	16	2.40	833	2.08	71
GAP2d	49.74	37038	1.86	6	10	1.75	248	1.96	6
GAP2e	51.06	38855	2.06	39	23	1.68	197	2.16	42
GAP3a	4.80	3045	2.31	3	27	1.65	158	2.37	3
GAP3b	(1000)	(308K)	1.98	11	16	1.64	131	2.01	11
GAP3c	56.25	35923	2.10	7	18	1.56	51	2.13	7
GAP3d	57.43	40139	2.09	8	19	1.52	33	2.16	8
GAP3e	3.18	1741	2.21	6	17	1.54	47	2.27	6
GAP4a	(1000)	(282K)	2.76	274	21	2.05	422	2.49	53
GAP4b	(1000)	(327K)	3.35	651	28	2.81	1031	2.58	112
GAP4c	(1000)	(322K)	2.32	15	21	1.64	125	2.42	15
GAP4d	2.36	825	2.63	8	27	1.55	56	2.72	8
GAP4e	(1000)	(283K)	2.10	50	17	2.03	431	2.12	8
GAP5a	39.39	24947	2.00	5	16	1.58	46	2.09	5
GAP5b	(1000)	(469K)	2.11	7	24	2.18	439	2.14	7
GAP5c	(1000)	(325K)	1.97	2	15	2.18	479	2.02	2
GAP5d	53.02	37773	1.83	0	12	1.61	61	1.89	0
GAP5e	(1000)	(440K)	2.01	49	20	3.35	1219	2.07	40

Table 12. Problems with less than 500 variables.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAP6a	314.45	109239	2.27	66	28	2.08	362	2.23	40
GAP6b	10.78	61.09	1.88	0	11	1.56	38	1.95	0
GAP6c	109.20	56435	2.26	6	25	1.64	73	2.39	6
GAP6d	65.98	33337	2.25	9	24	1.73	133	2.29	6
GAP6e	(1000)	(444K)	2.16	6	28	2.06	309	2.26	4
GAP7a	(1000)	(437K)	2.47	85	20	1.61	35	2.34	8
GAP7b	(1000)	(348K)	2.18	17	21	2.07	274	2.19	17
GAP7c	(1000)	(406K)	1.96	5	11	1.60	34	2.01	5
GAP7d	(1000)	(428K)	2.10	63	19	1.89	183	2.13	44
GAP7e	(1000)	(430K)	2.36	18	25	1.70	84	2.44	16
GAP8a	(1000)	(478K)	2.19	21	22	1.91	165	2.17	21
GAP8b	(1000)	(474K)	2.14	68	15	1.81	90	2.15	13
GAP8c	(1000)	(295K)	2.02	3	15	1.61	36	2.07	3
GAP8d	61.98	27735	2.06	3	13	1.61	20	2.14	3
GAP8e	(1000)	(410K)	2.13	31	21	1.83	102	2.25	26
GAP9a	(1000)	(421K)	2.13	62	21	16.40	6988	2.13	28
GAP9b	(1000)	(424K)	2.33	85	34	6.30	2426	2.42	102
GAP9c	(1000)	(370K)	2.07	2	24	1.86	174	2.13	2
GAP9d	(1000)	(422K)	2.10	13	27	2.27	366	2.16	11
GAP9e	138.70	62376	2.11	7	17	1.72	95	2.19	7
GAP10a	(1000)	(439K)	2.41	15	31	2.06	230	2.48	15
GAP10b	(1000)	(462K)	2.81	210	25	2.52	397	2.57	82
GAP10c	(1000)	(392K)	2.26	15	25	18.76	6804	2.29	15
GAP10d	(1000)	(327K)	2.17	26	24	2.10	214	2.27	21
GAP10e	(1000)	(426K)	2.09	9	21	2.54	359	2.18	9

Table 13. Problems with less than 500 variables.

C. RESULTS FOR PROBLEMS WITH 500 AND 600 VARIABLES

These problems began to challenge the techniques and the computer memory. In a few cases (GAPA1 and GAPD1), the time required to generate cuts is significant, causing solution time to increase. The large number of nodes required by ECB to solve GAP11c and GAP11e mark them as candidates for nested ECB. GAP12c and GAPD1 are the first two instances where the computer terminated the solution process prematurely. In both cases, there was insufficient memory to keep the extensive data required by the branch-and-bound process.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAP11a	172.81	59274	2.84	98	32	2.21	225	2.77	25
GAP11b	(1000)	(304K)	2.18	19	19	1.99	124	2.31	30
GAP11c	(1000)	(364K)	2.71	19	30	56.79	15423	2.70	19
GAP11d	830.71	192307	2.86	182	25	2.30	255	2.47	22
GAP11e	(1000)	(391K)	73.02	18892	43	61.26	18999	4.28	299
GAP12a	(1000)	(304K)	2.99	11	39	3.14	460	3.08	21
GAP12b	(1000)	(304K)	3.98	349	30	2.57	272	3.06	25
GAP12c	(983.62)	(304K)	3.20	43	34	4.52	810	3.26	34
GAP12d	(1000)	(304K)	3.56	36	43	2.91	390	3.56	35
GAP12e	(1000)	(304K)	2.34	11	26	1.76	44	2.52	22
GAPA1	190.48	64588	19.03	2609	24	1.89	109	10.24	122
GAPB1	(1000)	(311K)	9.16	519	27	2.50	363	7.81	88
GAPC1	(1000)	(326K)	17.34	3665	21	5.85	1172	6.65	416
GAPD1	(737.51)	(295K)	22.73	0	13	1.62	16	22.61	0

Table 14. Problems with 500 and 600 variables.

D. RESULTS FOR PROBLEMS WITH 1000 VARIABLES

Solving with knapsack cuts and branch and bound becomes increasingly difficult. ECB does well with the exception of GAPC3, which it cannot solve, making it a candidate for nested ECB. Cut generation time is significant, making ECB alone quicker in many cases, even when using more nodes in branch and bound. Two problems (GAPD2 and GAPD3) terminated early due to an unrecoverable failure during the branch-and-bound process.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAPA2	428.03	104332	82.19	1806	24	2.12	92	74.54	104
GAPB2	70.76	16845	45.38	1342	34	2.82	184	38.31	240
GAPC2	33.18	7692	33.64	1373	19	3.27	269	26.14	260
GAPD2	(506.52)	(126K)	484.44	8	29	1.84	20	482.71	8
GAPA3	(1000)	(221K)	18.79	866	53	2.22	102	14.44	160
GAPB3	(1000)	(261K)	40.84	5297	25	2.21	96	4.28	25
GAPC3	(1000)	(251K)	142.75	15681	56	(1000)	(162K)	20.07	1311
GAPD3	(805.84)	(207K)	19.69	8	39	1.95	43	19.75	8

Table 15. Problems with 1000 variables.

E. RESULTS FOR PROBLEMS WITH 2000 VARIABLES

Solving with knapsack cuts and branch and bound becomes still more difficult, and GAPB5 cannot be solved with knapsack cuts and branch and bound. ECB does well with the exception of GAPA5, which ECB cannot solve, and GAPC5, which requires over 20,000 nodes. Both are candidates for nested ECB. Again, cut generation time is significant. Two more problems (GAPC4 and GAPD5) terminated early due to an unrecoverable failure during the branch-and-bound process.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAPA4	40.07	5388	35.51	527	42	5.08	320	33.10	202
GAPB4	(1000)	(145K)	48.99	2990	25	3.35	147	11.77	127
GAPC4	(977.59)	(133K)	42.96	1760	43	3.91	224	24.76	37
GAPD4	661.91	94401	140.75	223	32	2.46	49	139.86	36
GAPA5	(1000)	(147K)	389.71	17879	90	(1000)	(89K)	293.70	9621
GAPB5	(1000)	(144K)	(1000)	(57K)	57	36.52	1913	44.13	1337
GAPC5	(1000)	(146K)	229.26	9787	75	277.36	22677	8.62	178
GAPD5	(949.12)	(139K)	6.46	30	43	2.82	95	6.40	15

Table 16. Problems with 2000 variables.

F. RESULTS FOR PROBLEMS WITH 4000 VARIABLES

Solving with knapsack cuts and branch and bound nearly impossible. GAPA6, GAPB6, and GAPC6 cannot be solved in 1000 CPU seconds with knapsack cuts. ECB does exceptionally well. Knapsack cuts and ECB combined take fewer nodes than ECB alone in every case, but more CPU time as cut generation time increases.

Model Name	BandB		KS			ECB		KS/ECB	
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Time	Nodes
GAPA6	(1000)	(76K)	(1000)	(56K)	64	9.03	454	33.52	387
GAPB6	880.84	66861	(1000)	(52K)	84	29.22	694	39.21	152
GAPC6	(1000)	(76K)	(1000)	(46K)	69	87.28	1244	39.09	199
GAPD6	(1000)	(75K)	47.17	125	62	4.21	109	47.53	66

Table 17. Problems with 4000 variables.

These problems are so large that the number of branch-and-bound nodes that can be transited

in 1000 CPU seconds is much fewer than the mid-sized problems, which possibly explains why there were no premature terminations of the solution process for these problems.

INITIAL DISTRIBUTION LIST

No. of Copies

1. Defense Technical Information Center.....2
   8725 John J. Klingman Rd., Ste. 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Dr. Richard S. Elster Code 01 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

4. Dr. Peter Purdue Code 08.....1
   Naval Postgraduate School
   Monterey, CA 93943-5002

5. Dr. Richard E. Rosenthal Code OR/Rl.....1
   Naval Postgraduate School
   Monterey, CA 93943-5002

6. Dr. R. Kevin Wood Code OR/Wd.....10
   Naval Postgraduate School
   Monterey, CA 93943-5002

7. Dr. Gerald G. Brown Code OR/Bw.....1
   Naval Postgraduate School
   Monterey, CA 93943-5002

8. Dr. Alan R. Washburn Code OR/Ws.....1
   Naval Postgraduate School
   Monterey, CA 93943-5002

9. Dr. Craig M. Rasmussen Code MA/Ra.....1
   Naval Postgraduate School
   Monterey, CA 93943-5002

10. Dr. Francois Melese Code 64/Me.....1
    Defense Resources Management Institute
    Naval Postgraduate School
    Monterey, CA 93943-5201

11. LTC Jeffrey A. Appleget 10
ATTN: MADN-A
Department of Mathematical Sciences
US Military Academy
West Point, NY 10996-1786

12. COL D. C. Arney 1
ATTN: MADN-A
Department of Mathematical Sciences
US Military Academy
West Point, NY 10996-1786

13. COL James L. Kays 1
Department of Systems Engineering
US Military Academy
West Point, NY 10996

14. Col Daniel W. Litwhiler 1
Department of Mathematical Sciences
2354 Fairchild Dr., Ste. 6D2A
USAF Academy, CO 80840-6252

15. Academic Research Division 1
ATTN: Dr. Stephen Landowne
US Military Academy
West Point, NY 10996-5000

16. Professor George B. Dantzig 1
EES & OR Department, MC: 4022
Terman Engineering Ctr., Rm 418
Stanford University
Stanford, CA 94305

17. Professor B. Curtis Eaves 1
EES & OR Department, MC: 4022
Terman Engineering Ctr., Rm 419
Stanford University
Stanford, CA 94305

18. Dr. Cynthia Phillips 1
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-1110

19. Dr. E. Andrew Boyd 1
Department of Industrial Engineering
Texas A&M University
College Station, TX 77843-3131

20. Dr. David Ryan 1
Engineering Science Dept.
University of Auckland
Private Bag
Auckland, New Zealand

21. Dr. David P. Morton 1
Department of Mechanical Engineering
Engineering Teaching Center
The University of Texas at Austin
Austin, TX 78712